5-1-2006

The BabyBot – Robotic Child Monitoring System

Christole Griffith
Indiana University - Purdue University Fort Wayne

Parul Reddy
Indiana University - Purdue University Fort Wayne

Table of Contents

Acknowledgments

We would like to thank our advisors, Dr. Yanfei Liu and Dr. Carlos Pomalaza – Ráez for their continual support and advice on the different phases of the project.

We would also like to thank Dr. Hosni Abu-Mulaweh the Senior Design Coordinator for his informative and structured guidance.

We would like to extend our thanks to Dr. Hossein Oloomi for sacrificing his free time to assist in the preliminary setup of this project.

Finally, we would like to acknowledge the Indiana-Purdue University Department of Engineering for providing the necessary equipment, such as the Khepera II robot, to complete this project.

Abstract

This project developed a robotic system that can assist parents in child monitoring. In this system, the robot is capable of finding and following a 7-10 month old baby (crawling age) in a confined space. In addition, this system also features some artificial intelligence design in order to determine the danger level that the child might be in and take actions accordingly. Various algorithms were generated to accomplish the tasks of target finding, object tracking and obstacle avoidance. Finally, a prototype of this system was generated and experimental tests were conducted. The prototype includes a Khepera II robot, named BabyBot, a scaled down model of a room with a total area of 32square feet. This room contains a small model of a baby, generated from yellow Lego blocks, and three obstacles created from small Lego pieces. With the equipped camera, BabyBot is capable of processing images to determine whether certain objects are present. BabyBot also displays some artificial intelligence capabilities such as activating distraction light circuit as well as an alarm warning when necessary.

SECTION I

Selected Design

Selected Design

Section 1.1: Introduction

The design of this project was classified into different sections and subsections that describe the various initializations that are required to complete the project. The environment setup will shed some light on the environment that will be built to simulate a baby's playpen area. The hardware design is broken down into the application of the different circuits and interfacing necessary for the robot to display the artificial intelligence capabilities. The robot set up is a brief description of all the turrets and their setup. The software design section describes the algorithm that will tackle the problem of finding the baby initially, depth perception to determine the child's location, and finally the algorithm for path planning with obstacle avoidance. Given below is a basic description of the layout that will follow:

- Environment setup
- Hardware design
    - Artificial Intelligence
        - Sharp GP2D02 IR Ranger distance measurement sensor
        - The use of lights as a visual distraction
        - Bread board parental alert alarm
- Robot setup
- Software design
    - Path Planning Algorithms
        - Cellular Automata algorithm
    - Finding the baby
        - Camera

Section 1.2: Environment Setup

The first task for this project will be to construct the environment in which the system will be tested. Given below in Fig. 1 is the final desired environmental output.

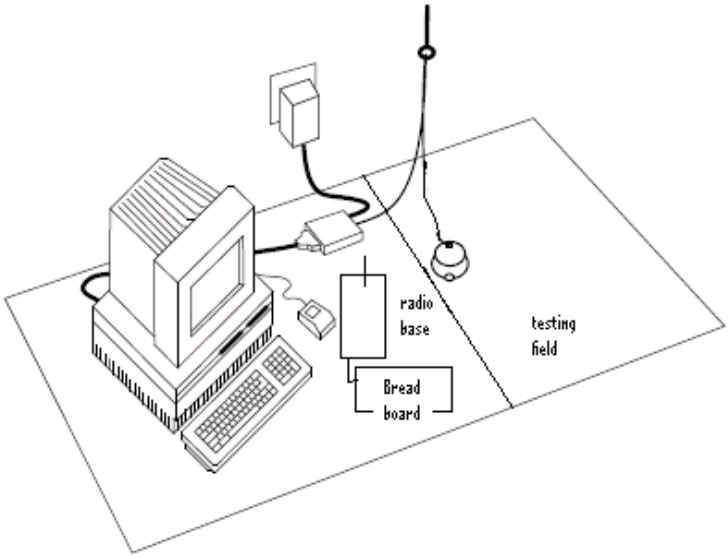


Figure 1: The robot and host computer configuration

General Environment

In order to accomplish this we propose to use a mobile robot to follow and monitor a child between the ages of 7 to 10 months (crawling age). The motion of a child, at this stage of development, is very unpredictable. Thus, we will incorporate some randomness into path generation of the child, based on how interested it is in the surrounding environment.

The testing will take place in ET 349, the robotic lab. This lab has no windows that allow natural sunlight to enter the room. Thus, all setup and implementation will be under average room lighting.

A testing field will be built to simulate a baby's playpen area. The field will be a table with approximate dimensions of 1.2 X 2.4m (4X8 feet). This will act as a scaled down model of a real area. The table surface will be smooth and white. Also, there will be a red marking to indicate the out of bounds area and a blue marking to indicate the warning area. The red marking encompass the playpen and has a total surface area of 1.3 square meters (14 sq.ft.). The blue marking will be 5cm to the inside of the red line and will also encompass the playpen area. For clarification of this configuration please refer to Fig.1. Additionally, to provide a wall the table will have a white vertical boundary, of height 0.3 m (1 ft), on its perimeter.

In order to maintain the scaling properties for this project the child will have smaller dimensions as well. The approximate height and width of the child for the duration of the simulation will be 45 mm with a speed of less than 1 m/s. Also, the model of the child will be yellow.

During the final stage of the testing up to three different Lego blocks will be used as obstacles that the robot will need to avoid. These blocks will differ by color and size as compared to the child. It is desired that the blocks are less than 42 mm in height. Also, as a precaution no objects or markings in the testing area can be black or very dark in color.

Given below in Fig.2 is a top view of the prototype environment that will be built to test the system. Path 1 is a random path that the child might take during the testing.

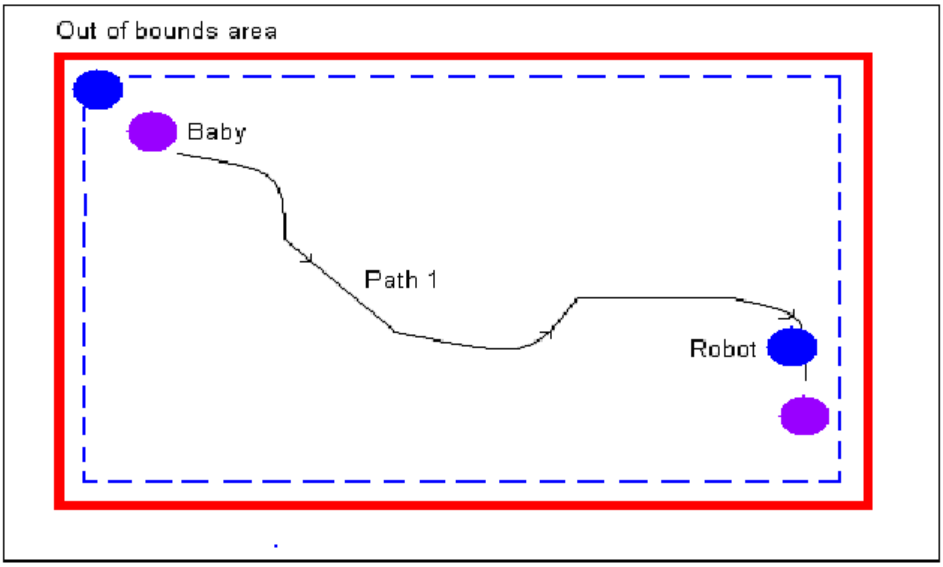


Figure 2: Prototype design environment

Host Computer Configuration

The configuration shown in Fig.16 will allow the robot to communicate with the host computer through a serial link. This will allow the computer to aid in the processing of image data. The length of the S serial cable is limited to two meters for proper operation. The robot will activate the distraction/alarm circuitry using a wireless transmission sent to the radio base. The host computer is linked to the interface/charger module (Appendix II) using a standard RS232 line. The interface/charger module converts the RS232 signal into S serial signal to communicate with the robot. The radio base receives a wireless signal from the robot to activate the proper circuit. In order to use the serial communication mode, the following connections will need to be made correctly. The robot must be connected to the interface/charger module using the S serial cable. This cable is also be used as a power source if needed. This external power supply is used when the general battery switch is OFF. If the switch is ON, the robot uses its own batteries as a power supply. The interface/charger module must be connected to the host computer using a standard RS232 cable. The interface/charger module must also be

connected to the AC/DC adaptor using the power supply jack. The encoding wheel on the robot sets the correct mode of operation for the robot to communicate with peripherals. Mode 9 (115200 bits/s) allows the robot to be controlled by a host computer using the serial communications protocol [1].

The transfer time of each image will depend on the size of the picture as well as the speed of the serial communication line. Each pixel will have two different parameters RG/GB (R – red, G – green, B- blue) that corresponds to the color of the pixel. Each pixel size will be multiplied by 2. Thus, each color will have its own matrix value that will be transferred to the computer.

$$Transfer\_time = \frac{x \times y \times B \times 2}{speed}$$

where,
x = number of pixels in the x direction = 160
y = number of pixels in the y direction =120
B = number of bits per pixel. Here it is assumed that each pixel has a size of 1byte. To convert this into bits we need to multiply it by 8.

$$B = 1 \times 8 = 8$$

Speed = speed of the serial line = 115200 bits/s

Therefore the $Transfer\_time = \frac{160 \times 120 \times 8 \times 2}{115200} = 2.6667\,\text{sec}$

This means that it will take about two and a half seconds to transfer an image from the robot to the computer using the Serial cable/RS232 cable. This shows that the RS232 link to the computer is a little slow as compared to other connections such as USB connections which have a speed of 480Mbits/sec. Modern robots are equipped with faster microprocessors that have greater memory storage. This allows the robots to process the data faster on their own microprocessors instead of sending it to a computer. Thus, the speed of the data transfer will be considered adequate for Baby Bot.

Section 1.3: Hardware Design

In order for the robot to display some artificial intelligence capabilities, additional components are required. These hardware components will enhance the robots' response to the different situations it might face. The names displayed in parenthesis are provided as reference. From now on these short names will be used to refer to the appropriate equipment.

List of Components
1 40 pin female socket (J1)
1 Breadboard (B1)
5 LEDs (red, yellow, or green) (LED1 – 5)
1 Voltage source/Power supply (VCC)
5 470 Ω resistors
5 10 kΩ resistors
1 1 kΩ resistor
1 100 kΩ resistor
1 22 kΩ resistor
1 100 μF capacitor
1 1000 μF capacitor
1 0.270 μF capacitor
1 0.1 μF capacitor
1 0.068 μF capacitor
2 1N4001 diode
5 2N2222 transistors (Q1 – 5)
1 2N3702 transistor
1 4-pin JST connector (J2)
1 Sharp GP2D02 IR Ranger distance measurement sensor (Sen1)
1 Nutone Chime kit

During the initial setup, the radio base will be attached to B1 with the help of the J1 connector. The voltage source is also attached to B1 and set to an initial value of 5 V. Fig. 3 depicts the setup for the circuit and a detailed explanation of the lighting process.

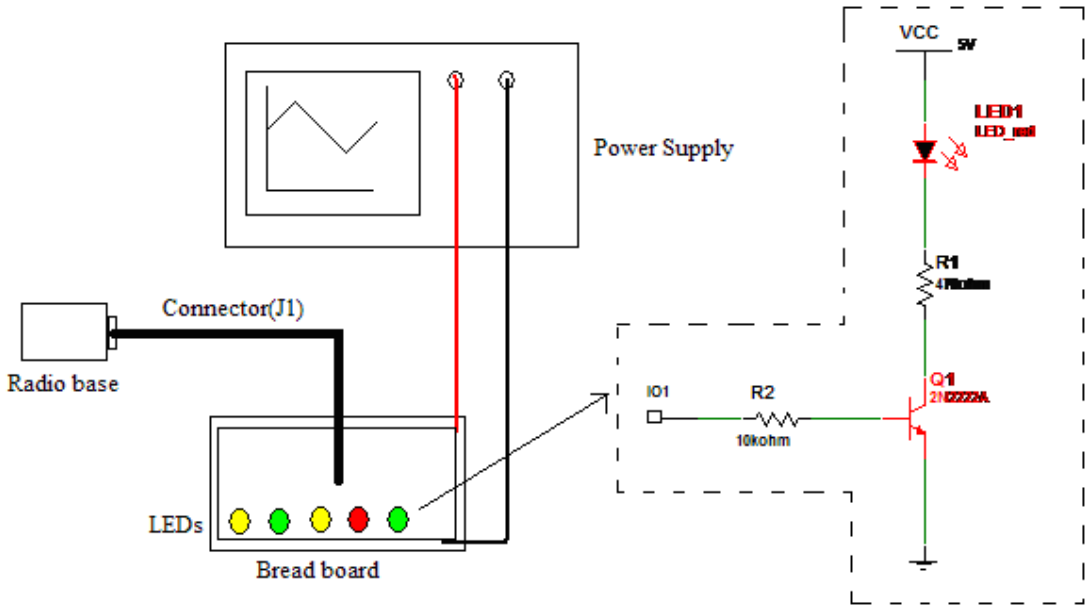


Figure 3: Initial setup – hardware components

If the baby is approaching the out-of bounds line or about to enter into a dangerous situation, then the robot will decide to turn on the LED circuit. In order to do so it will first have to determine if the baby is in any imminent danger. If so, the robot will transmit an RF signal to the radio base. Once a signal is received the processor on the radio base will determine to send a signal through J1 to B1.

Information provided in the dotted lines of Fig. 3 is the specific detail of the circuit connections to be made on the bread board. Once the I/O pin goes high, current will flow through R2 and to the base pin on Q1 down into the collector. Since there is a small voltage drop from base to collector (V_{bc}) the transistor will start conducting. This means that the current will begin to flow from VCC to ground. During this process the LED will light up as desired. This circuit will be replicated in order to have five LED's light up.

The current through R2(transistor base current) is given by,

$$ib = \frac{Vbb - Vbe}{R2}$$

where,
ib = base current
Vbb = base to ground voltage. Here it is assumed that once the I/O pin goes high Vbb will become 1V.
Vbe = base to emitter voltage. For a typical NPN transistor the nominal value is 0.7 V
R2 = 10kΩ

$$ib = \frac{1.0V - 0.7V}{10k\Omega} \approx 30\mu A$$

$$ic = \frac{Vcc - Vce}{R1}$$

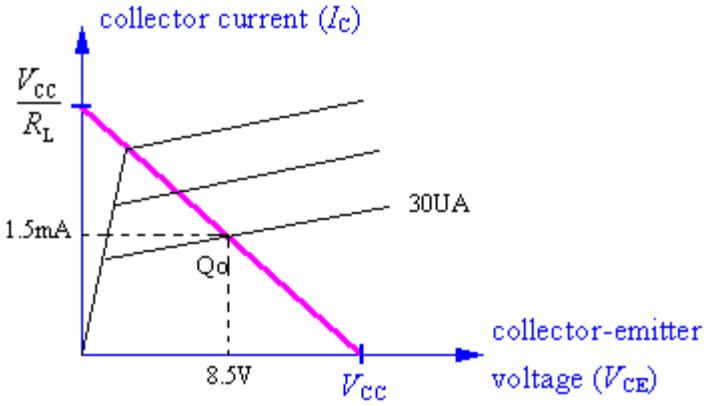


Figure 3a: Transistor V-I characteristics

From Fig 3a it has been determined that,
Vce = 8.5V and
$ic = 1.5mA$

A similar initial setup is required to sound the alarm. The purpose of having this alarm is to alert the parents that a child is in the out of bounds area. This way the parents have enough time to respond, before the child is seriously harmed. To implement this alarm module the circuit shown in Fig.4 will be used. Again, the I/O pin 1 from the processor present in the radio base will activate the alarm when required. The alarm used is a door bell kit that comes with a solid chime, one lighted push button and a 16V transformer.

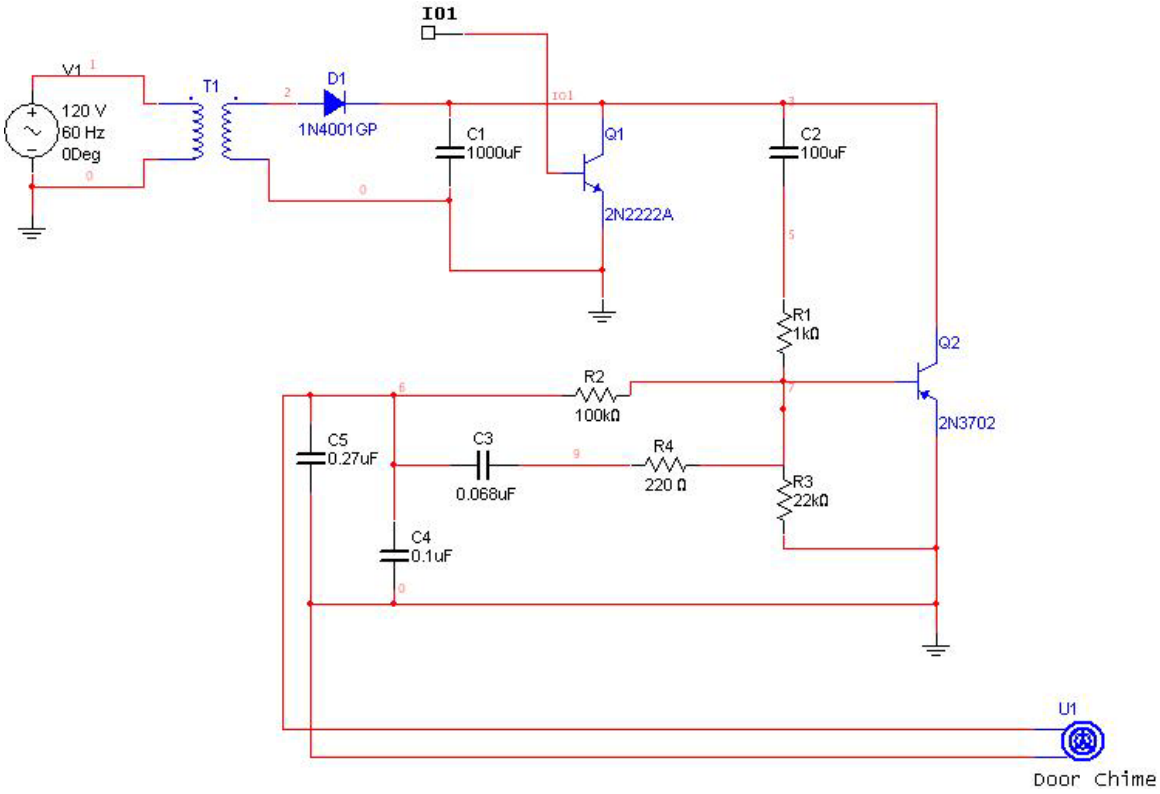


Figure 4: Alarm system

The components of the chime will have to be removed from the case to be installed onto the breadboard. The transformer will be connected to a 120V AC voltage source. Here also, Q1 acts as a switch that activates or deactivates the rest of the circuit. D1 is present as a protection diode that regulates the flow of current in one particular direction. The resistor and capacitor values for this circuit were obtained from the Birdie alarm. These values might be changed to suit our requirements during the testing stage.

Although the robot comes equipped with eight IR proximity sensors they might not be adequate for this project. The effective sensitivity of these sensors reduces after 100mm rendering them ineffective in maintaining a particular distance from the child. Thus, to keep a safe distance of 11cm from the child, Sen1 will be used. This sensor produces a non-linear voltage output that is inversely proportional to the distance of an object that is being tracked.
J2 (included in sensor kit, see Fig. 5) will be used to integrate the sensor to the robot.

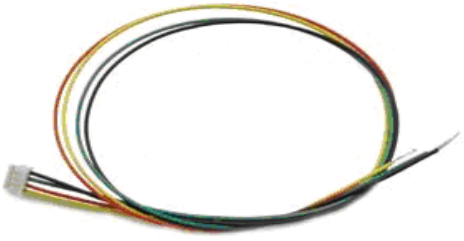


Figure 5: 4-pin sensor header

The four wires are connected to VCC, ground, clock input and data output. Fig. 6 illustrates the different connections that are required. In order to incorporate this sensor into the system the robot's general purpose I/O turret will be attached. Then a resistor divider circuit and diode will be connected in series to regulate the amount of voltage connected to the sensor. Its output will be fed back into the robot allowing the robot to estimate the distance between itself and the child.

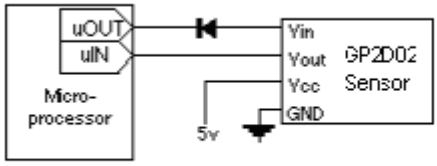


Figure 6: Block diagram of sharp sensor (Sen1) interface

All values of components and sensors are subject to change as more meticulous tests have been run on the robot and if the results are not as expected.

Section 1.4: Robot setup

The robot that will be used for this project is the Khepera II [1]. This robot is compact and has powerful microcontroller capabilities. For this design the robot will be assembled with the following turrets, mounted in the given order, from bottom to top: Base unit, General I/O turret, Radio turret and the Camera turret. To attach these turrets to the robot base they have to be placed on the extension connector with all the pins seated correctly. Then, gentle pressure is added to insert the turret into the extension connector.

Base unit

An overview of the base unit is shown below in Fig. 7

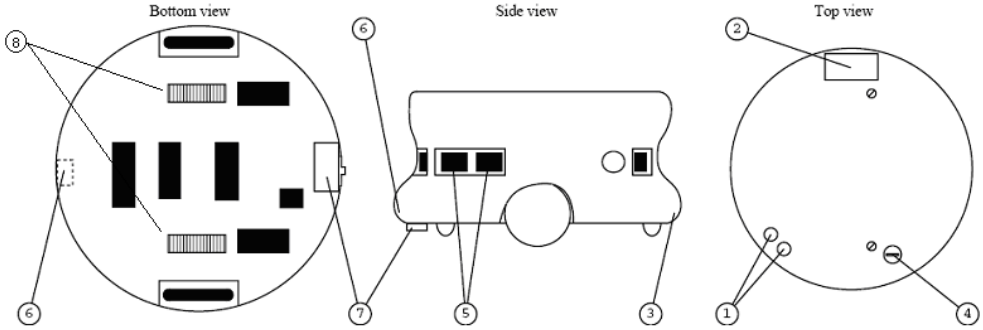


Figure 7: Base unit

1. LEDs
2. Serial line (S) connector
3. Reset button
4. Encoding wheel to select running mode
5. Infrared proximity sensors
6. Battery charger connector
7. ON/OFF battery switch
8. Wheels

General I/O Turret

The next component to be added will be the General I/O turret shown below in Fig. 8. As mentioned before this turret will be used to incorporate Sen1 into the system.

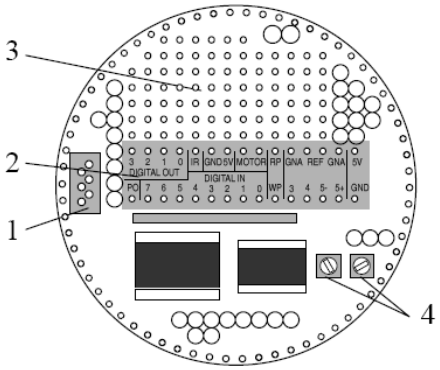


Figure 8: Overview of the turret layout

Components of the I/O turret:
1. Serial line (S) connector
2. I/O connections area
3. Free connections area
4. Analog input gain regulation

Radio Turret

An overview of the radio turret is shown below in Fig. 9. The serial connection of this turret will be used to communicate with the host computer.

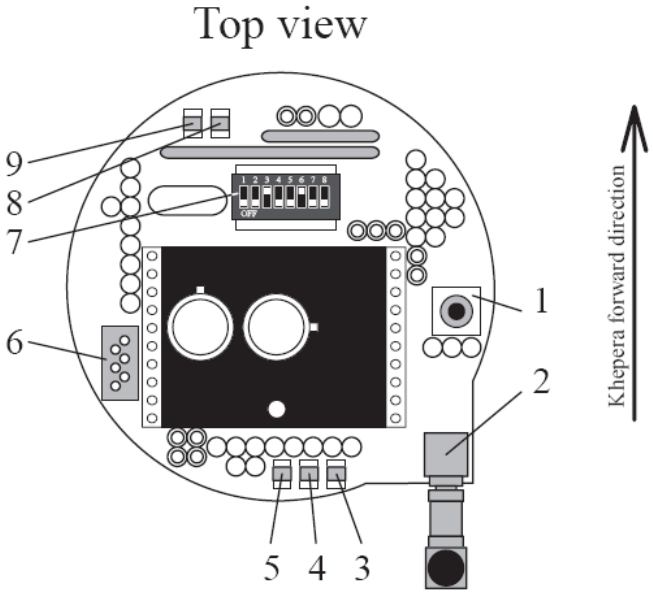


Figure 9: Overview of the turret layout

1. Reset button
2. Connector with antenna (antenna can be removed and turned)
3. Rx LED. When ON, the turret is ready to receive data.
4. Tx LED. When ON, the turret is transmitting data.
5. Carrier detect. When ON, the turret is detecting an active radio channel.
6. Serial line (S) connector.
7. Radio turret running mode and ID selector.
8. Repeat data LED. When ON, indicates that data has been lost and is repeated.
9. Lost data LED. When ON, indicates that data has been repeated 10 times without success and is considered as lost.

Camera Turret

Fig. 10 shown below gives an overview of the K6300 Camera Turret

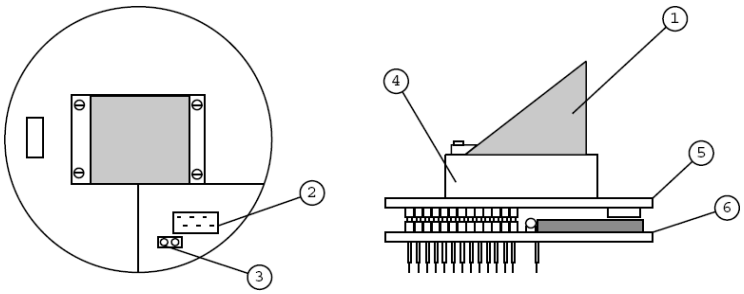


Figure 10: Camera turret

1. Optical prism
2. Serial connector
3. Jumper for stand alone mode
4. Lens case
5. K6300 vision board
6. K6300 processor board

The camera turret comes equipped with a MC68331 processor, along with flash memory and a reset button. Other components on this board are jumpers for stand alone mode, running mode jumpers and a serial connector that allows data to be transferred directly from the camera turret to a computer. If this connector on the camera turret were to be used, another serial connection with the host computer is needed. But the serial connector on the camera turret will not be used at this time. This is because the interface/charger module is equipped with only one S serial line connection. Thus, it would not be possible to control the robot and transfer images at the same time. Instead, the images will be sent to the radio base using the robot's internal bus connection. Then the image file will be sent to the host computer via the S serial line.

Section 1.5: Software Design

For the completion of this project it is necessary to divide the software design into 4 portions. This section will give a brief overview of the total algorithm process. Also, a more detailed description of finding the baby, vision based depth perception and the cellular automata path planning algorithm will be given.

Algorithm flow chart

The flow chart shown below in Fig. 11 gives a graphic explanation of the algorithms that will be generated in order to complete final product of Baby Bot.

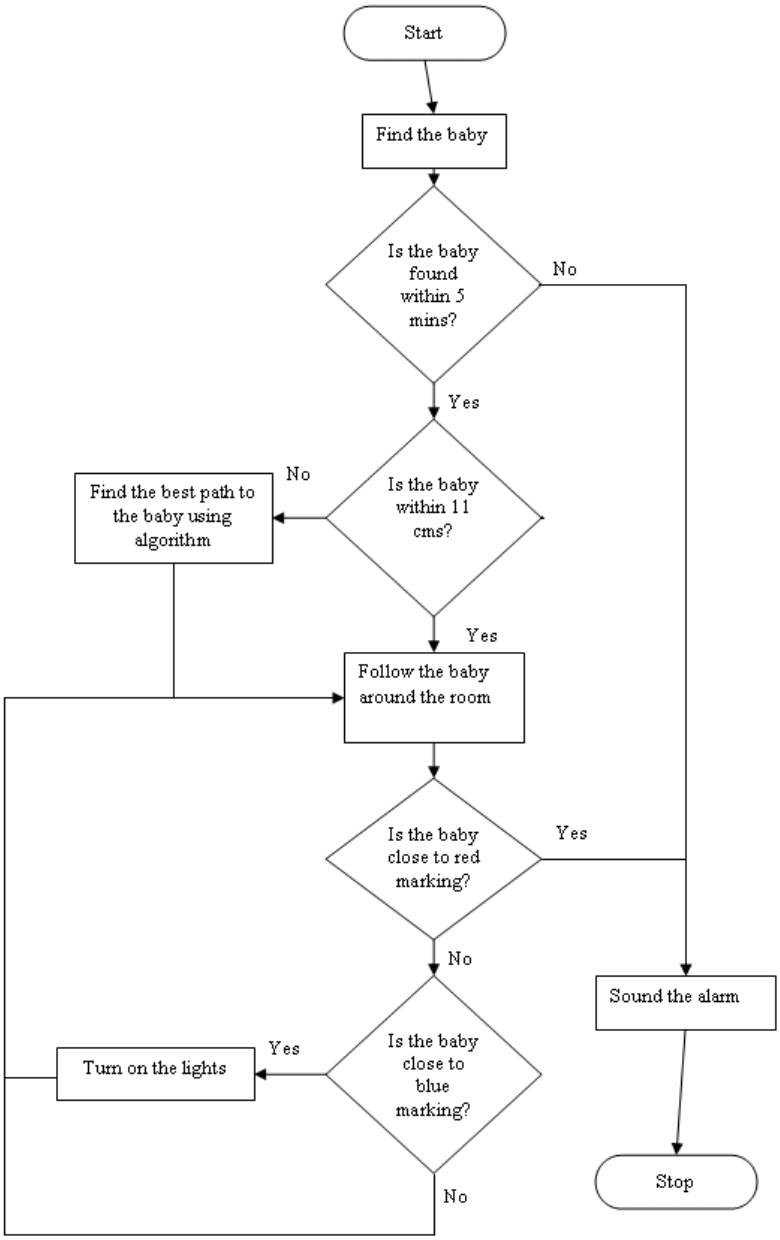


Figure 11: Flowchart of project

Finding the Baby

The camera turret of the robot will be used to find the baby. Once an image is taken the algorithm will check the color index of each pixel. Each line of the picture will be read until a yellow pixel is found. Once a yellow pixel is found a virtual flag will be activated. The robot's software application module contains a set of 64 system flags that can be read and changed from either a Lab View program or a turret user application. These flags can be used to set up a communication between the robot and the turret. The robot can read and change the flags using certain commands. The turret application can also read and change the flags using a built in function. For this project one of the flags will be activated when the required yellow pixel depicting the baby is found in an image. Then using serial communications protocol the robot will check the value of the flag. Once this flag is activated the process of finding the baby is complete. If the baby is not found, the robot will continue on the search path. Given below is a simple decision making algorithm skeleton that will be used to program the robot to find the baby.

Acquire an image
Read each line to determine whether a pixel is yellow
    If any pixel is yellow activate a flag
Read flag value to see if it is activated
    If flag has been activated, baby is found, go to next algorithm
    Else keep finding the baby

The next step, after the baby is found, is to get within the proper distance of the child

Vision based depth perception

If the child is found and Sen1 does not have a reading on the distance to the child the vision based depth perception will come into effect. The raw image data is not adequate to give useful information to identify the objects in the image. The depth perception from an image will be accomplished with the help of Lab View processing.

This processing includes the use of the threshold based method which employs high pass filter magnification of the object boundaries in the image data. This process makes it easier to detect the edges of an object. For this method a mask of [-1 -1 1 1] array is applied to the rising edge of the data from the image. Also, a mask of [1 1 -1 -1] array is applied to the falling edge of image data. This will create a correlation between the top view and the robots' perspective image as shown in Fig. 12. The computer will have a preexisting top view image of the complete environment, including the location of the obstacles.

The output of the application of the masks to all of the image data points is zero or a small number when the image is flat and becomes a relatively high number in case edges are present in the image. These edges are the boundaries of the objects in the image and are represented as peaks in the filtered image. So a rising edge, which is represented by a

positive peak, shows the starting point of an object, and the falling edge which is represented by a negative peak symbolizes the end point of the object. As the color difference between the objects become higher, the height and sharpness of the peaks between them also becomes higher. After the detection of the edges, a threshold is applied over the detected edges to eliminate the false edges, caused by the noise.

This will allow the robot to know the starting and ending points of the objects in the image. The host computer will have a preexisting image of the testing field with which this new image will be compared. As a result, the baby's position relative to the obstacles on the field is determined. Then the computer will send a signal to the robot to tell it where to move to be within the required range from the child [5]. Thus, the robot will have an idea of where the baby is located as compared to the other obstacles on the table.

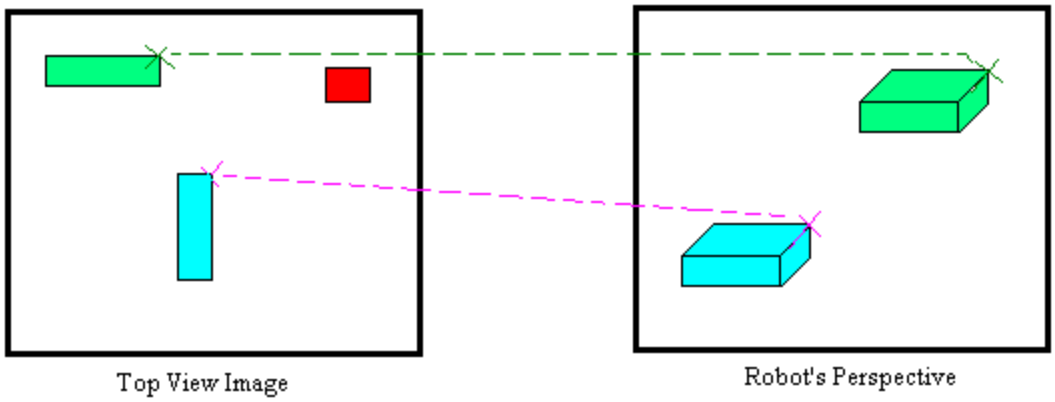


Figure 12: Correlation between different image perspectives

Cellular Automata algorithm

The robot will try to move to a cell adjacent to the goal cell. For this method the robot needs to initially capture a picture of the entire area in question. Then the image is processed into small cells. Each cell is given a number based on the following:

1-Free cell
2-Obstacle
3-Goal
4-Start

Thus, once the space has been divided onto a grid the robot will then find the shortest distance to the goal while maneuvering itself around the obstacles. To do this an adjacency graph will be built. The nodes of this graph will be free cells and the edges will be connections between two free cells. Therefore, the robot will never intersect with a cell that contains an obstacle. It will always move to the midpoint of a free cell. From this adjacency graph the best path will be chosen to reach the goal. Given below in Fig. 13 is a sample path taken by the robot to get close to the child.

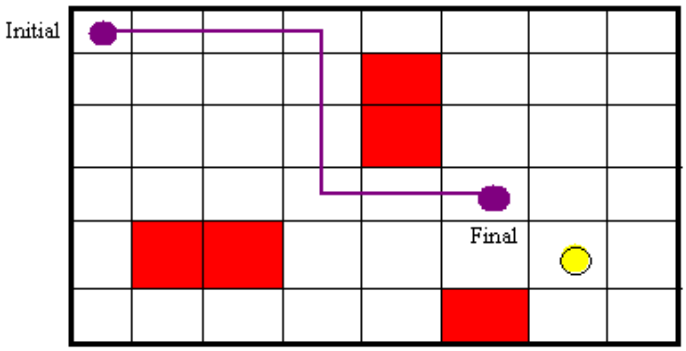


Figure 13: Sample path planning output

Conclusion

This design was the result of extensive research done on the individual aspects of this project. However, robotics is a fast growing technology where newer models and components are being released. Due to this, better components and algorithms might be available in the future. As more research is done to obtain a better understanding of the internal workings of the robot, changes might be made to optimize the performance of the Baby Bot.

SECTION II

Building Process and Changes

Running Mode Set-Up

Section 2.1: Procedures

In order to ensure proper communication and operation of the robot the following modes should be set prior to the start of any testing. These modes will determine the speed at which data is transferred between the robot, its turrets and the computer. Following the table is a detailed description of all the steps necessary to set each of the required modes.

Running mode selections

Base unit	Mode 8	57600(bits/s)
Radio Turret	Mode 4	Extension Turret
Camera Turret	Mode 3	KT-Grab requirement

Base Unit

To set the running mode on the base unit turn the encoding wheel to 8 as shown in the top-view image of the base unit in Fig. 14. Please note that the robot is only marked with every other running mode. In order to best determine the mode one must, using the small screwdriver provided, start at zero then count the number of clicks that the wheel makes as it is turned in the clock wise direction.

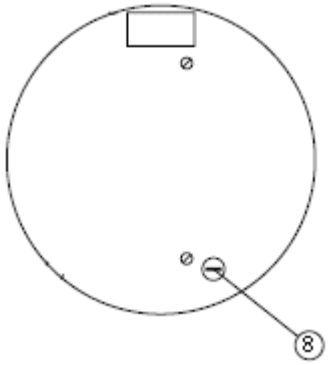


Figure 14: Encoding wheel for mode 8

Radio Turret

The radio turret is used as a simple slave turret for this project so the following selector settings are to be made. Each pin of the selector works as an ON or OFF switch and with the OFF side of the switch marked. This selector is located in the middle of the radio turret. Fig 15 shows the correct positions of the selector switch that sets the turret in Mode 4.

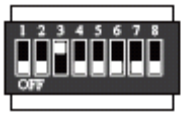


Figure 15: Selector Switch for Mode 4

Camera Turret

Serial communication mode 3 was used for the camera turret. The jumpers shown below in Fig. 16 are located on the turrets microprocessor board under the prism board. To set the running mode to 3, place the metal jumper in the two outermost spots as indicated in the right side of Fig. 16 below.

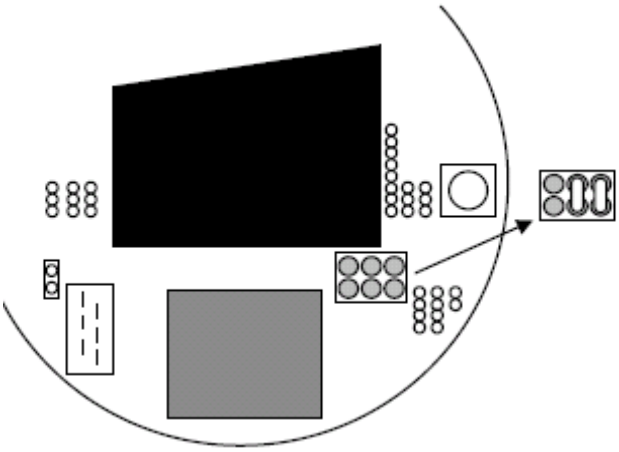


Figure 16: Jumper positions for mode 3

Using KTProject

Section 2.2 Introduction

The KTProject folder allows for the storage of the necessary files needed for the computer to communicate with the robot. This is a graphical C development for Windows that allows for writing C programs to control the Khepera II robot. The cross compiler allows for total control of the robot functions, such as motor control and sensor reading. It also allows for communication between the robot and additional turrets. The BIOS manual contains the commands that can be used to generate complex multitasking functions.

Get the Software

- The first software to obtain is the KTProject environment
  - Go to www.k-team.com and select the following links
    - Support Header
    - Download
    - Khepera II
    - Click on the link "KTProject environment for Windows"

- Install the KTProject software by following the setup instructions
- Write or download C program files and a Makefile
  - Do not put any spaces in the folder or file names – Spaces may cause an error when the KTProject is running

- Open KTProject software, the opening screen is shown below in Fig. 17

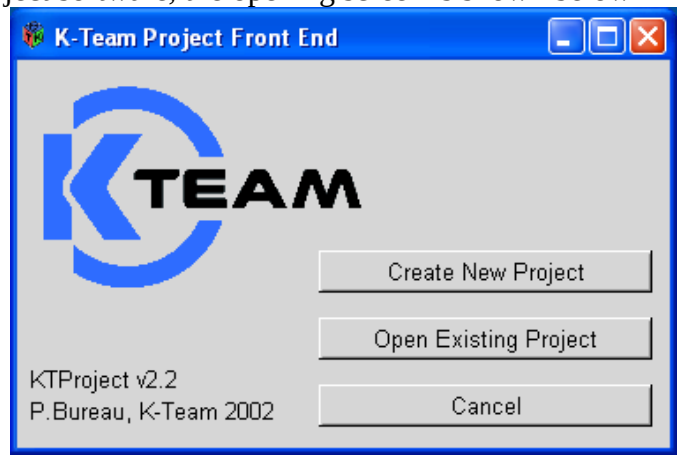


Figure 17: Opening screen of KTProject

Creating a new project

- From the opening screen of KTProject select “Create New Project”. Once clicked the result should be as shown in Fig. 18 below.

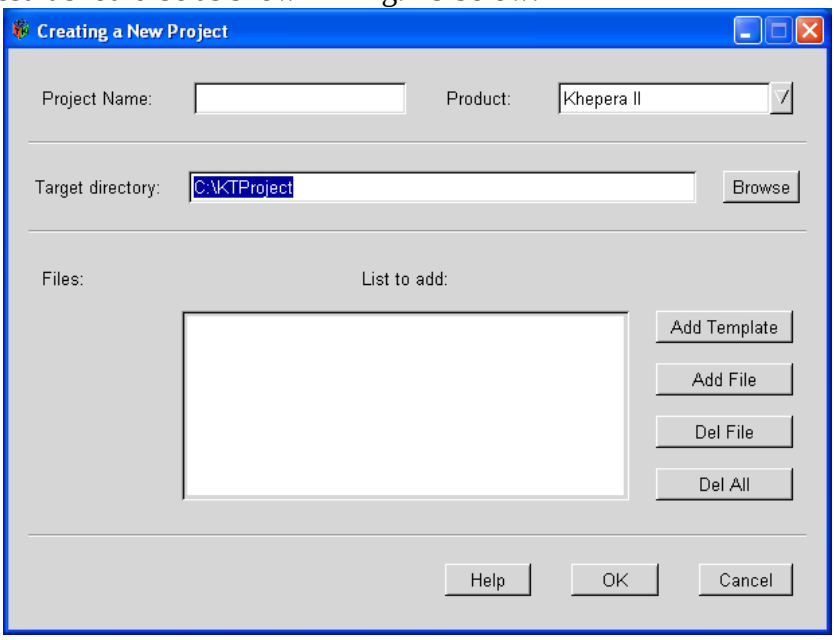


Figure 18: Creating a new project

  - Use the Browser to find the folder and filenames of the required C files and makefiles or type in the path (be careful of spaces)
  - Add the C files (filename.c) and Makefiles in this window by highlighting them and selecting the “Add File” button (When saving files the C drive usually works best)
  - Next Select the “OK” button

Open an exiting project

- From the KTProject Front End Screen (Fig XX) select “Open Exiting Project”. The result is shown below in Fig. 19

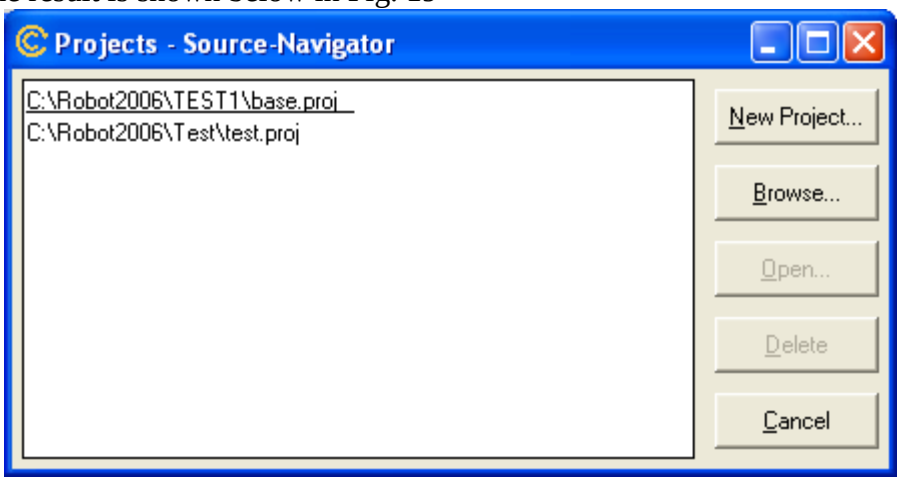


Figure 19: Source navigator

- Select the project to open by clicking on it once
    - Once the project has been selected the “Open” button will become available. This is the next button to be selected
- If the required file is not listed, click on “Browse..” and locate the file from the drive

Given below in Figs. 20 and 21 are examples of the added files. These files can then be modified to the users’ requirements.

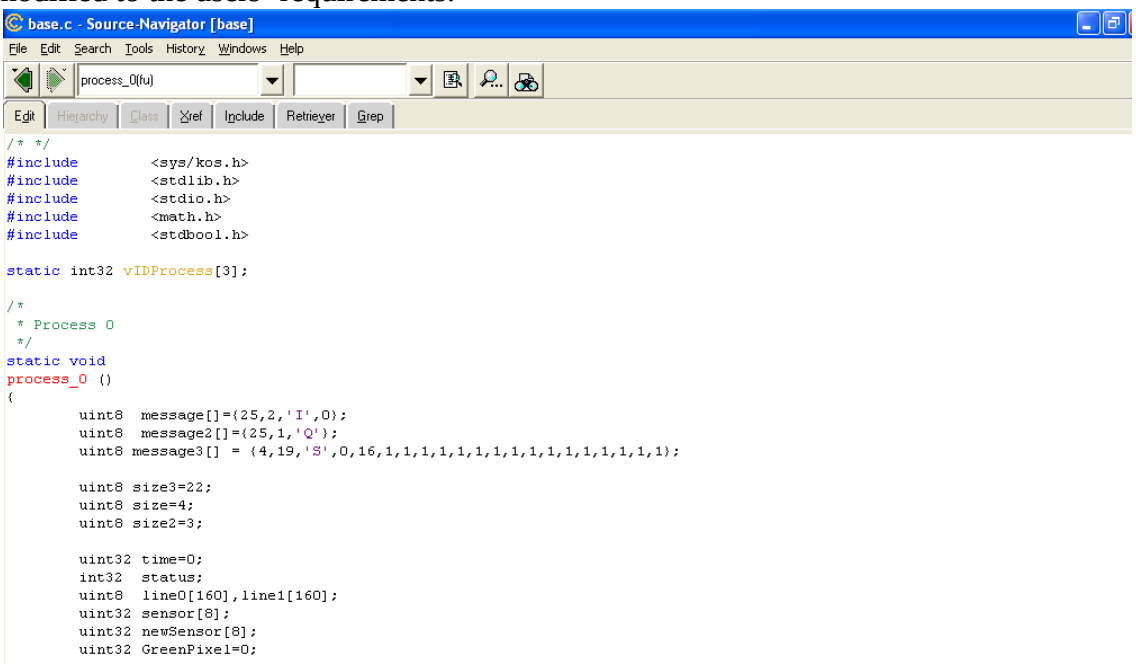


Figure 20: C code as view by the source navigator

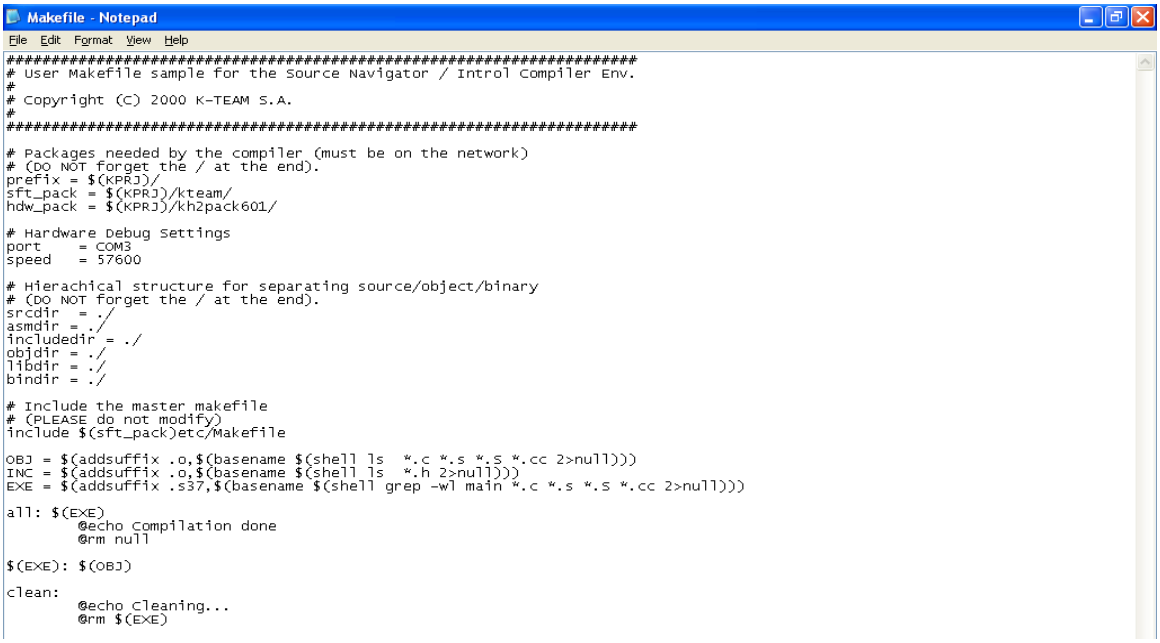


Figure 21: Makefile example

Building After Creating or Opening a Project

- After the changes have been made, the project needs to be built
  - In the Source Navigator, while viewing the C file, select the Tools Menu
  - From that menu select “Build”
  - Click “Start” to build the project

The confirmation, if the project has been successfully built without errors, is shown in Fig. 22.

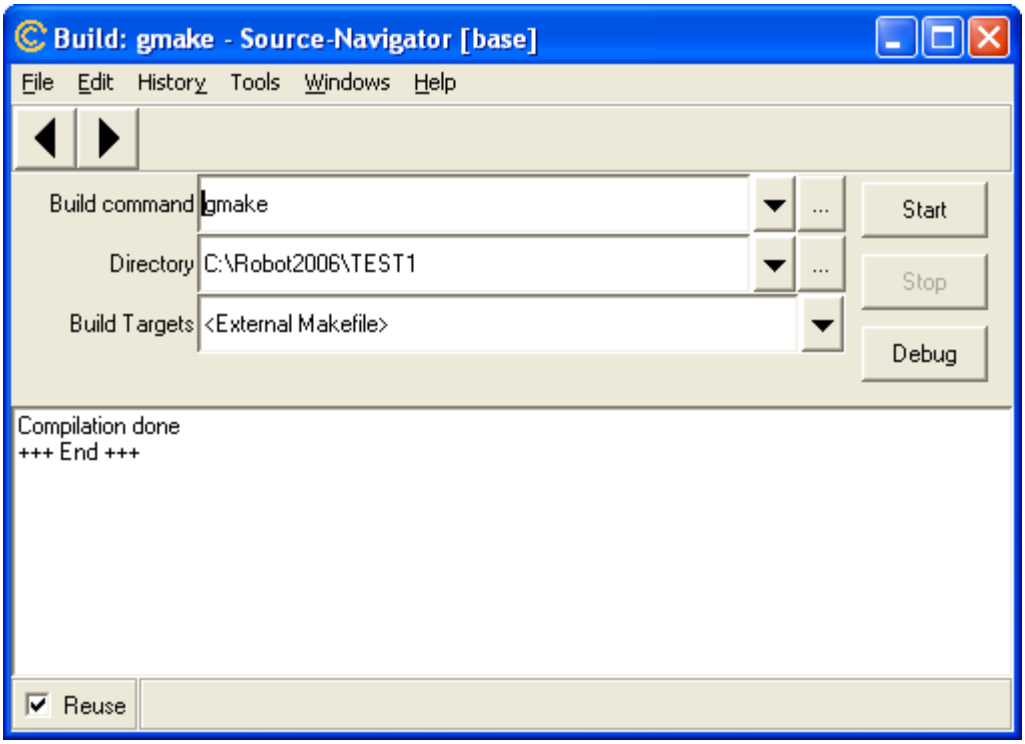


Figure 22: Confirmation build is complete

- Please note that in the folder where the project was saved there will be the new file with the same file name as the C file you created, such as, “filename.s37”

Serial Link Setup

Section 2.3: Setting up the Serial link

- Download a terminal emulator program such as (Tera Term, http://hp.vector.co.jp/authors/VA002416/teraterm.html)
- Install the emulator program by following setup instructions
- Open the terminal emulator program by double clicking on the icon

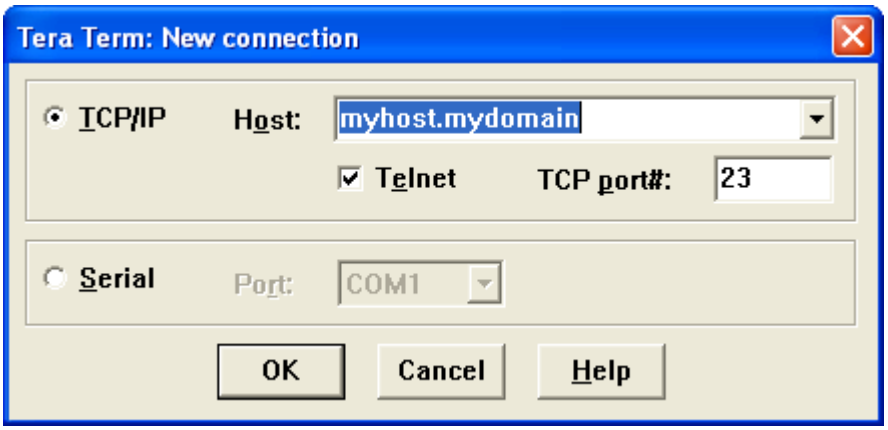


Figure 23: New connection screen

- In Fig. 23 above Choose "Serial"
  - Once the Serial tab has been selected choose the COM port that is used to connect to the robot
  - Then select "OK"
- Under the Setup Menu
  - Next select "Terminal setup"
  - Highlight the local echo as shown below in Fig. 24 so that the writing on the terminal emulator screen is visible

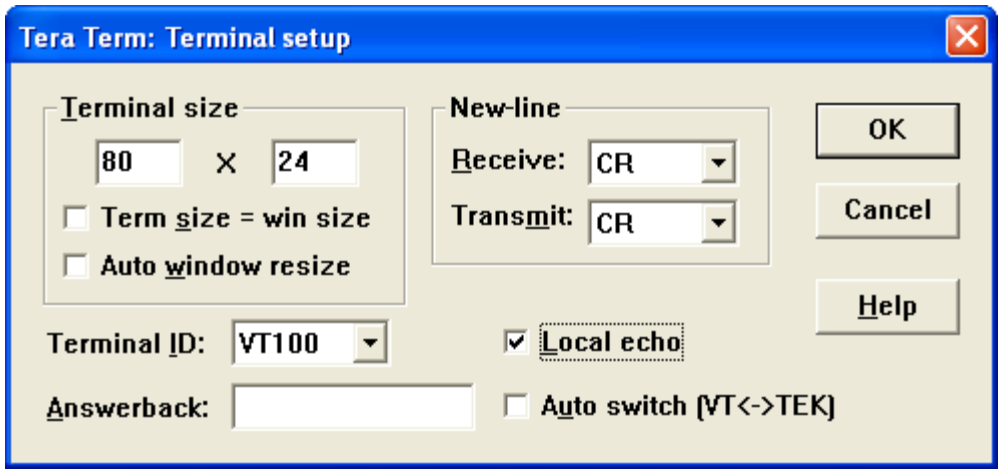


Figure 24: Local echo

- Under the Setup Menu select “Serial port setup”
  - Make sure all the parameters are as they are desired. Fig 25 shows the requirements used for this project.

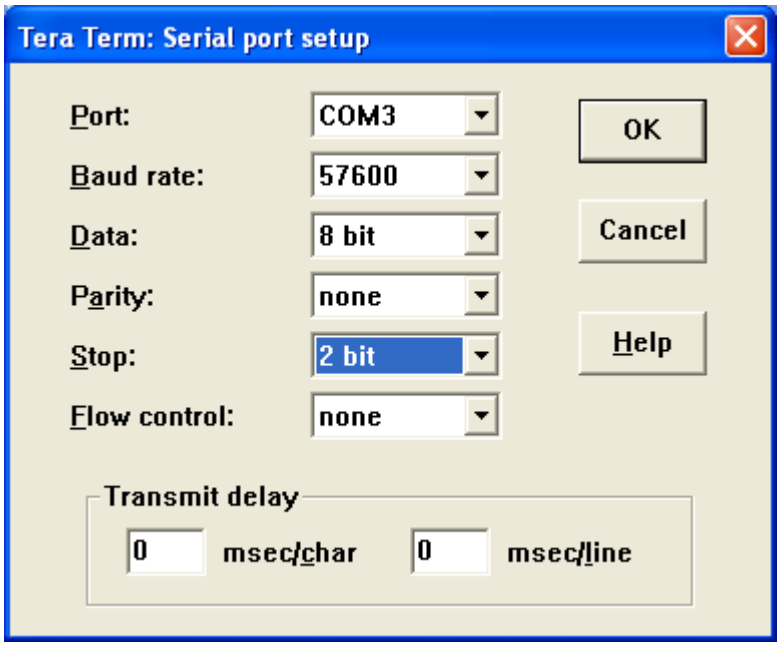


Figure 25: Serial port setup

It is possible to save the setup but there can be problems with reopening the link. Therefore, it is best to minimize the terminal emulator at this time so that the setup steps do not have to be repeated in the near future.

Section 2.4: Serial Communication Mode

First connect the robot to the computer as described in the Host Computer Configuration on page 8.

- There are several way to start the communication process
  - Plug the power cord to the interface/charger module
  - Push the reset button on the back of the base unit
  - And sometimes it possible to type “restart” in the terminal window
  - The response should be as shown below in Fig. 26

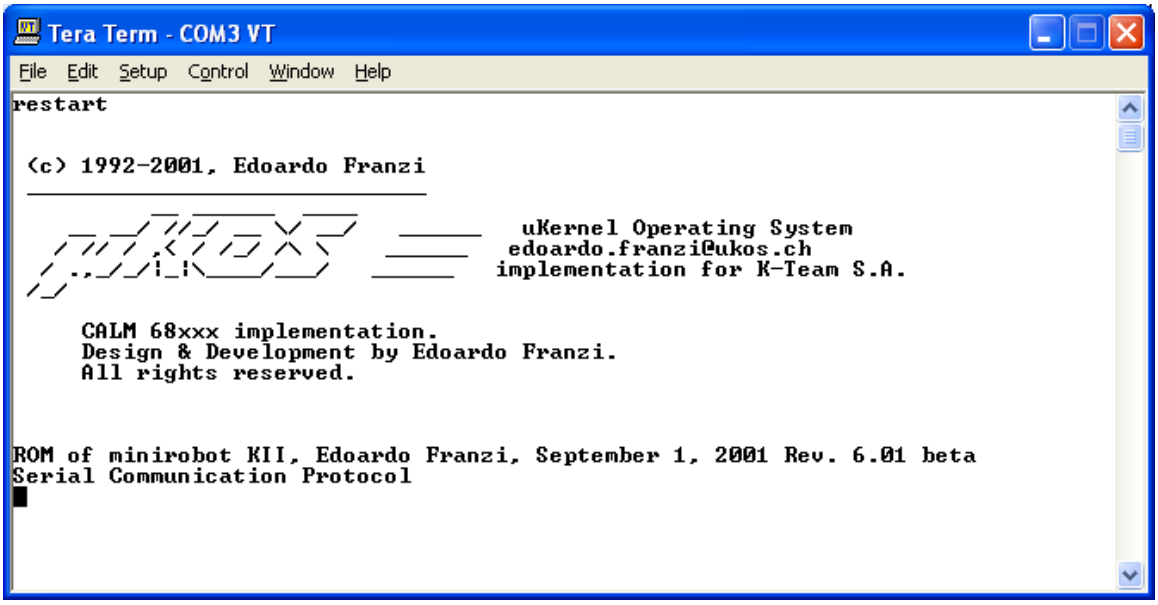


Figure 26: Serial communication protocol confirmation

- Check to see that all the turrets you have hooked up are connected properly by typing “net”
  - Depending on the turret in use the response will vary. For this project the correct response is shown below in Fig. 27

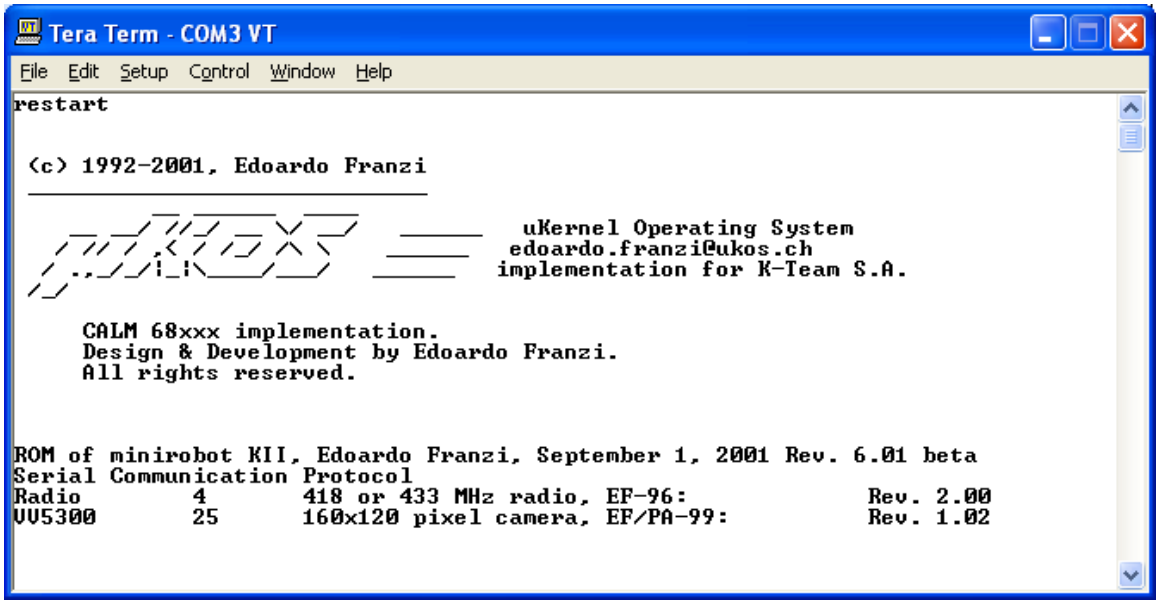


Figure 27: "net" command response

Testing the Link

Now that the setup is complete, the following set of simple instruction will allow for the link to be tested. For a more complete list of commands please refer to Appendix C.

Base Unit

- The base unit does not have a turret ID
  - To get a sensor reading from all 8 base unit IR sensors:
    - Type "N" followed by carriage return (Enter ↵)
    - Shown below in Fig. 28 is an example of the response

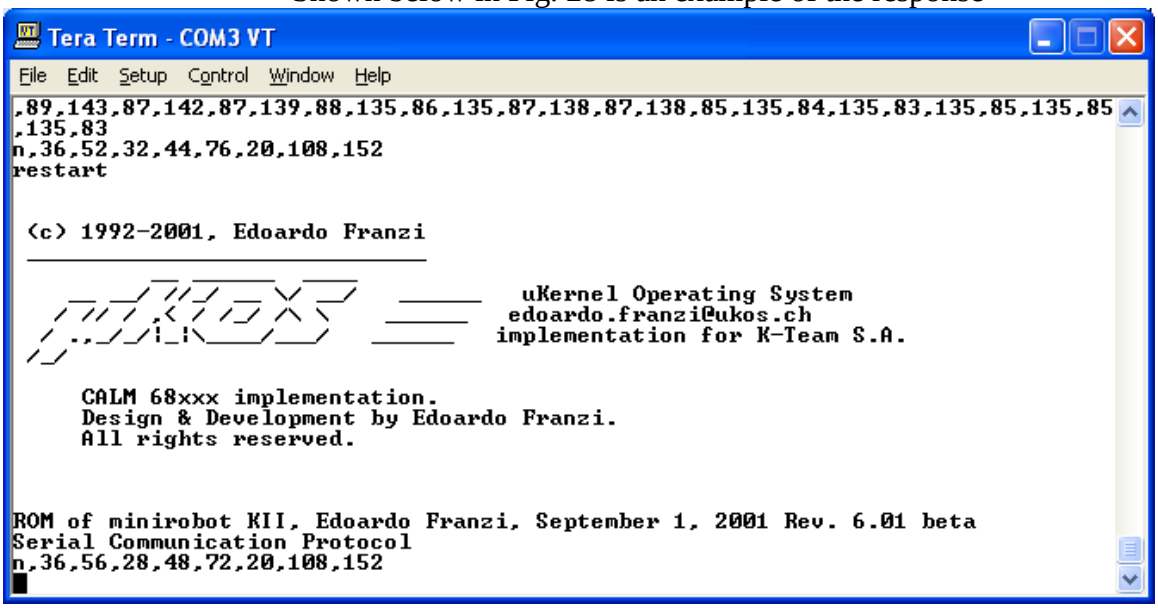


Figure 28: IR Sensor response

Camera

- The turret ID for the Camera is 25.
- To take a picture and read the line zero of that picture:
  - Inside the terminal emulator type
    - "T,25,Q" followed by carriage return
    - "T,25,I,0" followed by carriage return
    - The correct response is show below in Fig. 29

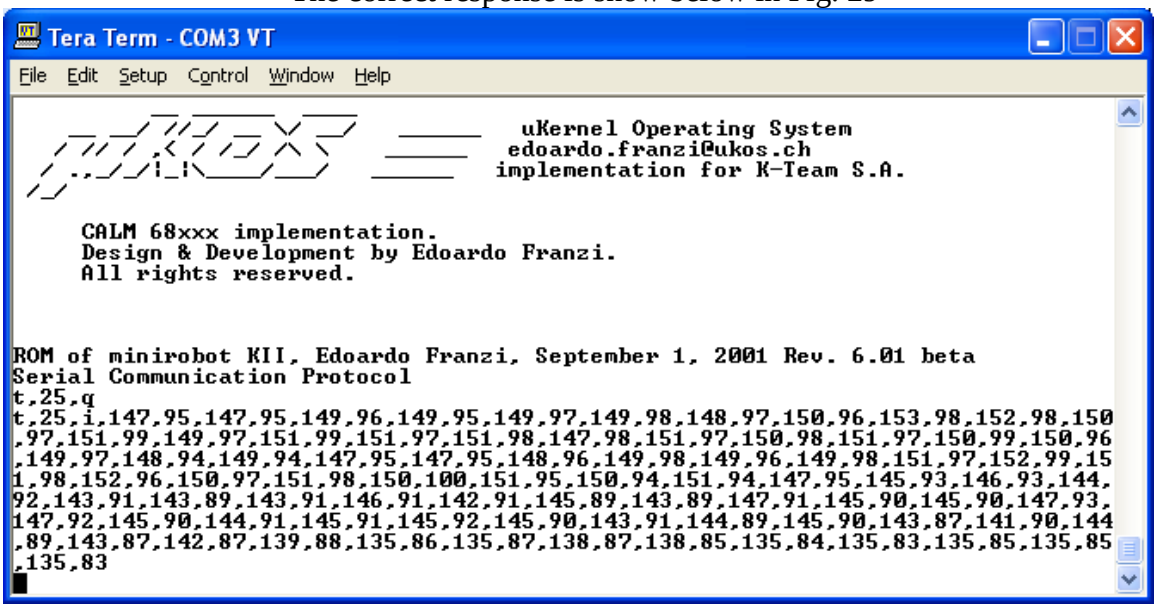


Figure 29: Camera turret response

Radio Turret

- The turret ID for the Radio is 4
- To Send a message to the radio base:
  - Inside the terminal emulator type
    - "T,4,S,0,16,1,2,3,4,5,6,7,8,9,0,1,2,3,4,5,6" followed by enter
    - "The correct response is shown below in Fig. 30

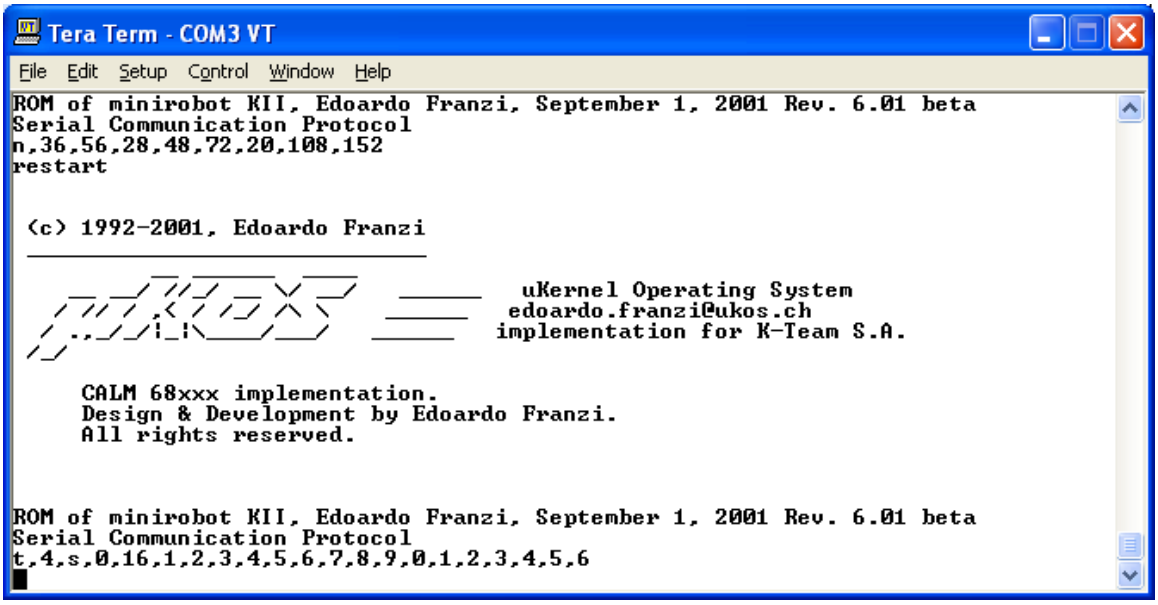


Figure 30: Radio turret response

Downloading a file to the robot

- First the S Loader must be started
  - In the terminal window type “run sloader”
  - The response is shown below in Fig. 31

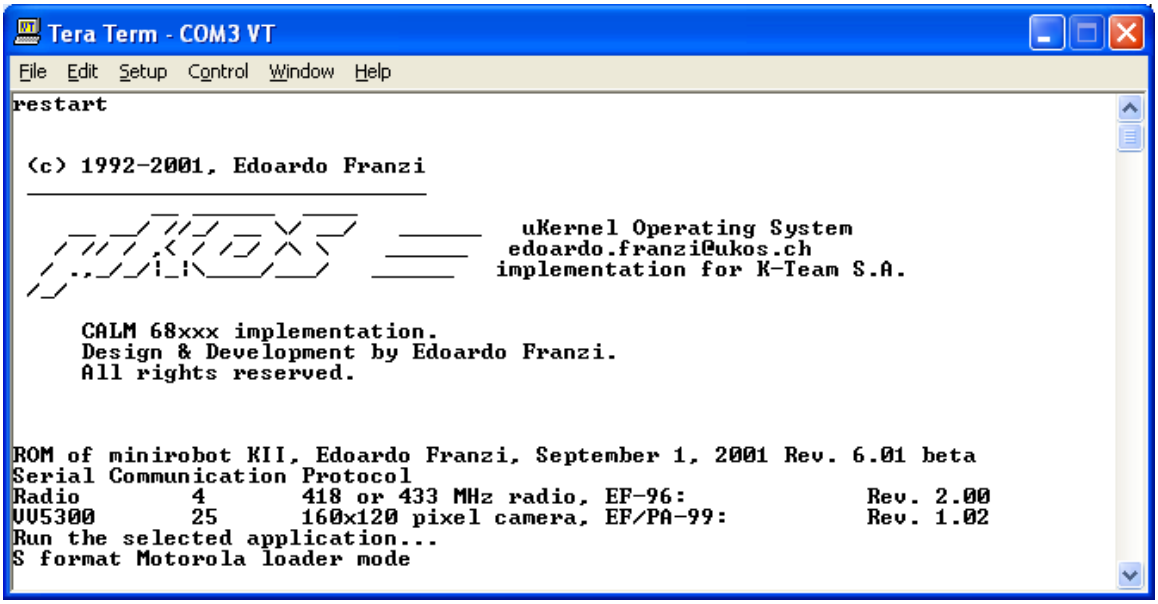


Figure 31: S loader response

- Now under the File Menu in the terminal emulator
  - Select “Send file”, Fig.32 shown below should be what is displayed next

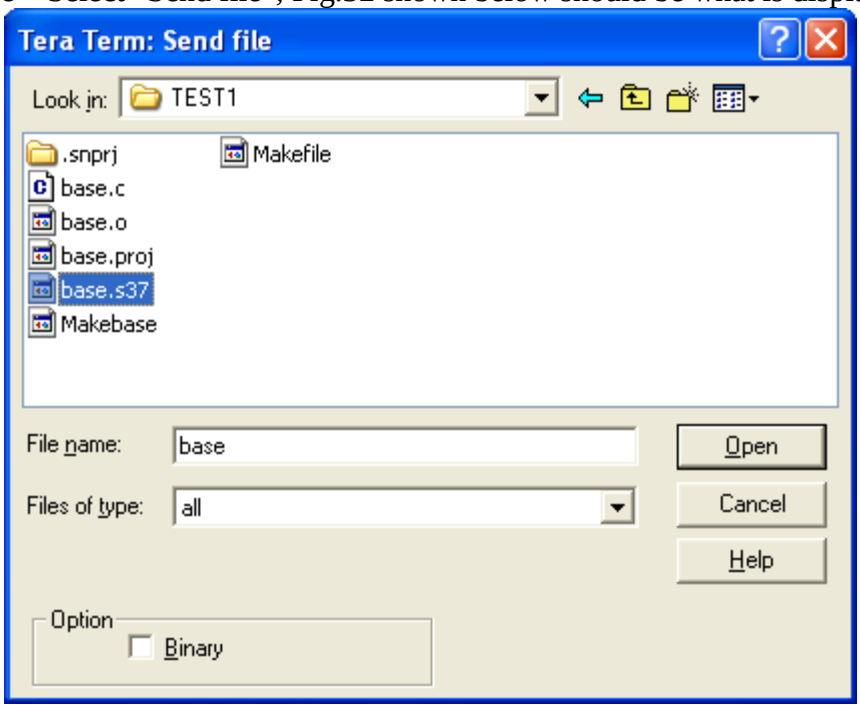


Figure 32: Sending file

  - Use the browser to locate the project folder that was created earlier
    - Inside the project folder select the “filename.s37”
    - This file should have been created when the C file was built. If this file does not exist please refer to the section on Building

After the program has finished downloading the robot will start to execute it. Please remember to hold the serial cable up off of the table so it does not impede the movements of the robot.

KTGrab

Section 2.5: KTGrab

The KTGrab software can be used to obtain a viewable image from the camera turret. The serial cable must be connected directly to the turret as shown below in Fig. 33. This means that information would not be transferable between the base unit and the host computer during operation. The emulator window should be closed before using KTGrab. Therefore, there would be no way to display the actions of the robot in the terminal emulator. It would be possible to use the KTGrab software and the serial connection to the robot base at the same time but a second interface/charger model would be needed.

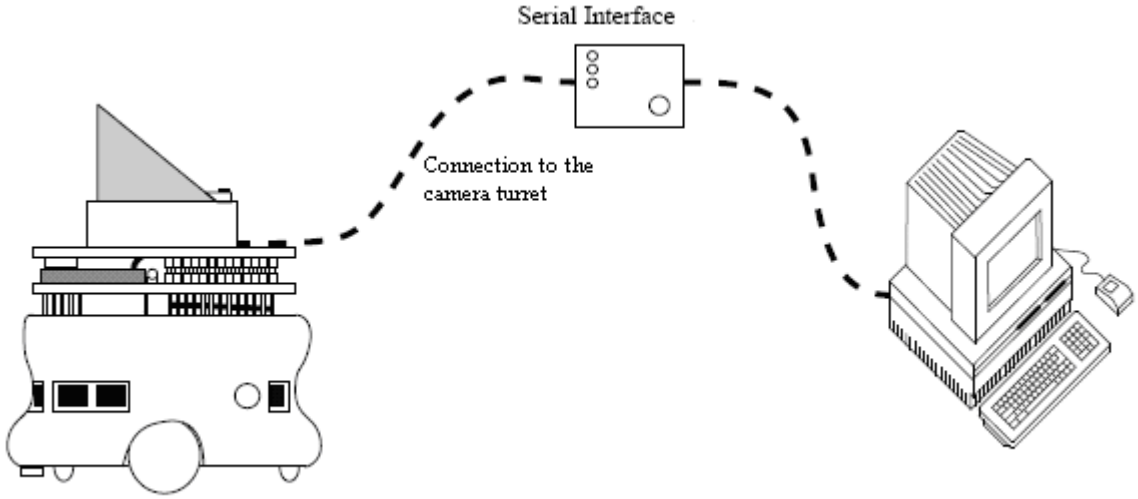


Figure 33: Camera turret interface

Download the Software

- To download this software for free
  - Go to www.k-team.com
    - Support Header
    - Download
    - Khepera II
    - Click on "KTGrab Executable for Windows"

- Install the software by following the setup instructions

- Open KTGrab software, the setup screen is shown below in Fig. 34 a,b

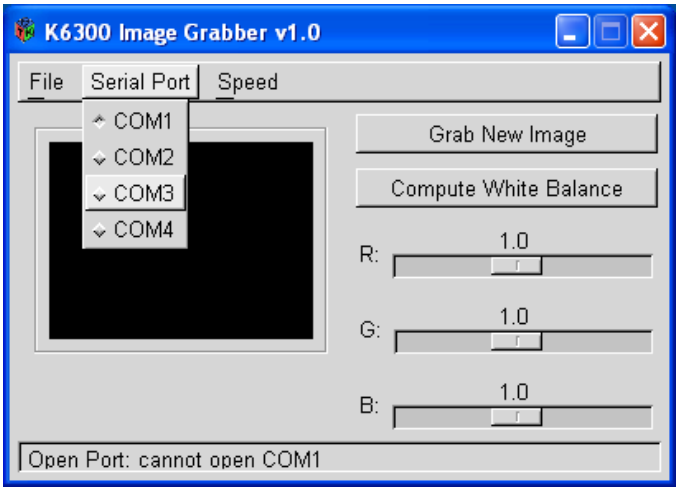


Figure 34a: Serial selection

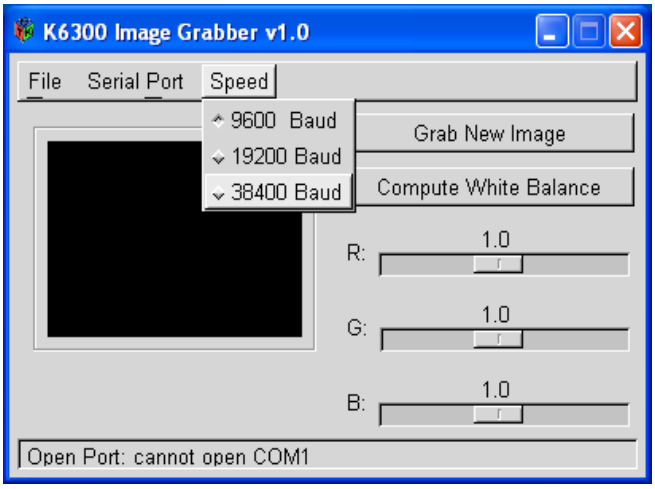


Figure 34b: Speed selection

- First select the Serial Port Header (Fig. 34 a)
  - Choose the COM port that the camera turret is connected to
- Than select the Speed Header (Fig. 34 b)
  - The camera turret should already be set to running mode 3 at this point so choose 38400 Baud
- Now that the setup is complete click on the "Grab New Image" button
  - The response will be similar to the image shown in Fig. 35 below

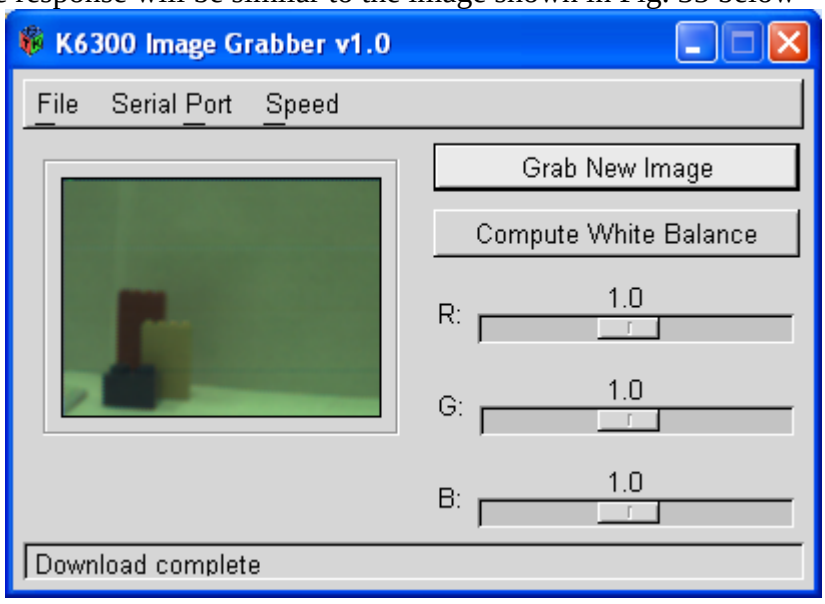


Figure 35: Sample KT-Grab image conformation

Environmental Changes

Section 2.6: Environmental Changes
Location

Last semester it was proposed that the testing would take place in ET349. But due to the lack of space in that lab all testing and setup took place in the ET324 lab instead. Since this lab does not have any windows either, all testing was performed under average room lighting. Given below in Fig. 36 is the testing area showing some of the major components.

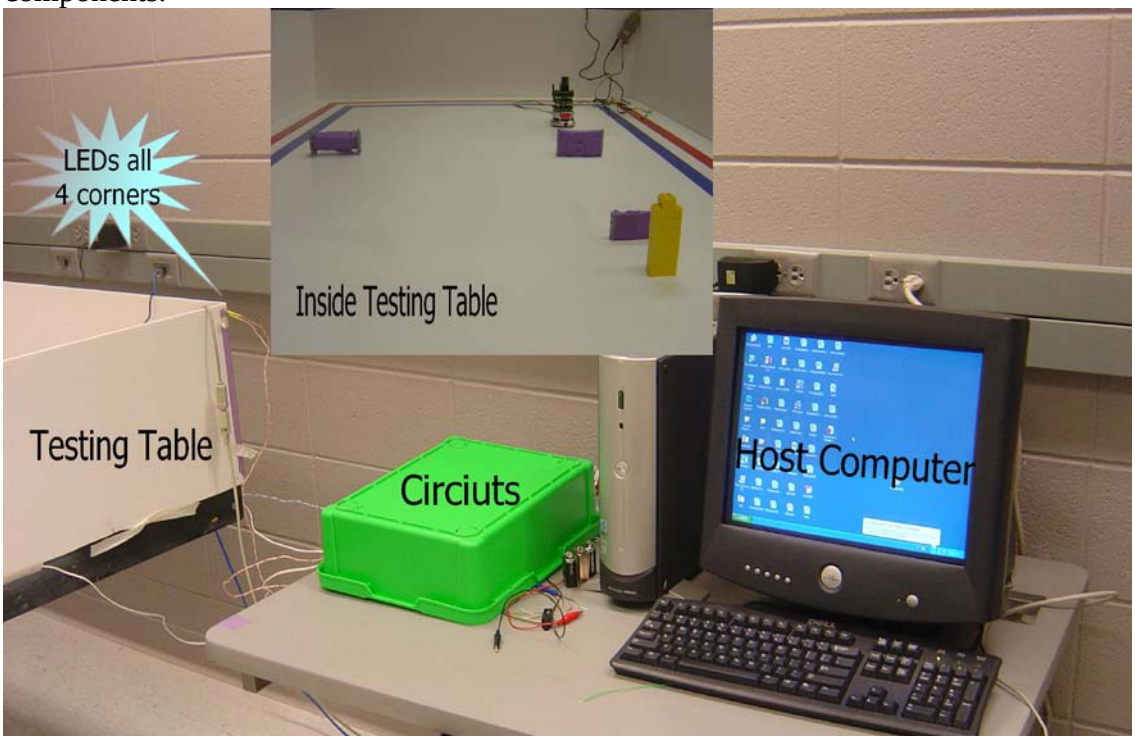


Figure 36: Testing Area

Dimensions

With the addition of all the necessary turrets over the base of the robot, the height of the robot changed to about 10cm. The camera turret is the final addition atop all the turrets therefore the extra height altered the camera's field of view. Therefore, to ensure that the robot is able to view the baby using the camera from as close as10cm, the dimensions of the baby were changed. The baby was created by stacking up eight Lego blocks and is 3cm long, 1.5cm wide and 7.5cm high.

Obstacles

While testing, it was determined that the robot could not differentiate between certain Lego blocks and the general environment. If the Lego blocks were either red or blue, the

robot assumed it was the markings on the table. This could provide negative results if the robot assumed the line was actually a Lego block or vice versa. Due to the poor quality of the camera the white background looks green from the robot's point of view. This is due to lack of white balance present in the raw data provided by the camera processor. Hence green Lego blocks were out of the question as well. Other dark colors, such as grey or black were not a plausible alternative either. The infrared (IR) sensors on the robot, which are used for distance estimation, do not detect these objects since the IR beam does not reflect properly from them.

After extensive testing it was determined that it was possible to detect purple since it could be easily differentiated from the general environment. This is the reason the obstacles are purple. They are composed of six plastic rods held together with small Lego parts. Two of the obstacles are rectangular box shaped. The third obstacle is cylindrical in shape with two Lego gears on either side holding the rods in place. . The dimensions of the obstacle are given below in the obstacle reference table. Additionally all 3 obstacles are picture below in Fig. 37.

Obstacle reference table

Obstacle #	Length (cm)	Height (cm)	Width (cm)
1	7.5	3	0.9
2	10	5	Diameter 4
3	10	5	0.9

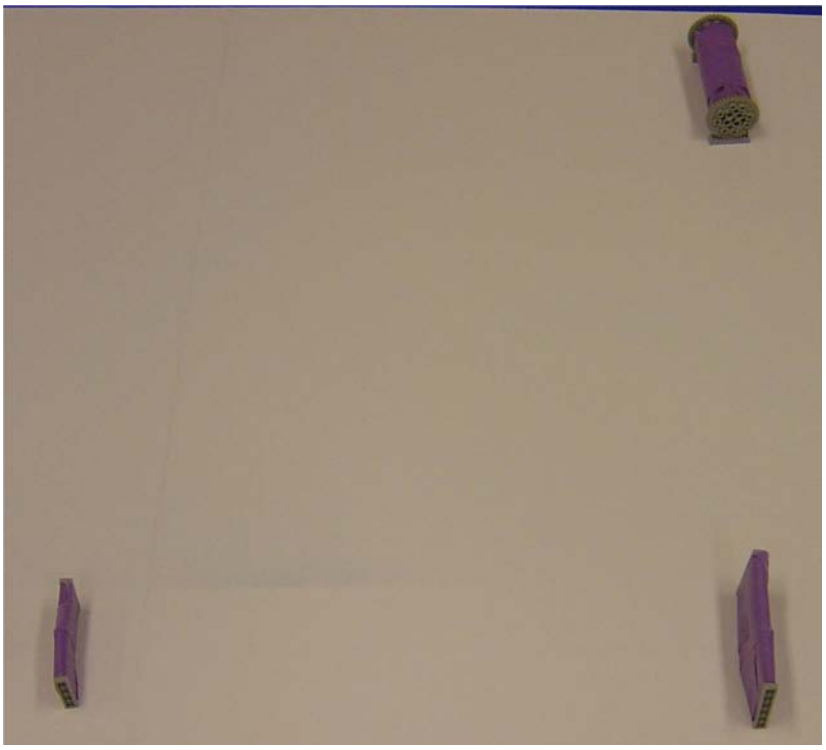


Figure 37: Obstacles

Hardware Modifications

Section 2.7: Hardware Modifications
Timer Circuit

After the robot sends a signal to the radio base, the same signal is instantaneously then transmitted out from the RS232 port present on the radio base. This means that the signal goes high for only a few milliseconds. This is not enough time to turn on a transistor and activate necessary hardware components. So, in order to extend the time of the input signal to a few seconds a 555 timer IC (integrated circuit) was used. This circuit is positive edge triggered and is activated when pin 3 of the RS232 goes high. The output of the timer IC is long enough to trigger the transistors.

Alternating Circuit

Last semester it was assumed that while sending a signal through the radio base, different signals could be sent to activate different circuits. For example, a signal to activate the alarm would be different, maybe in content or size, than one sent to activate the lights circuit. But after testing the radio base component it was confirmed that it was not possible to decode the information because of the setup of this project. The reason for the presence of the RS232 port on the radio base is to attach it to the computer and receive information from it. Then the decoding process could take place. Since this is not a viable option for this project an alternative method was necessary to switch between the two circuits. Although there are drawbacks to this design at least it was possible to replicate the switching using hardware. This method requires no external manual work by the robot or human.

Bell Circuit

The information provided by the manufacturers of the Nutone Chime Kit did not provide any specifications of the transformer that was used to run the bell. This is the reason a generic design was chosen last semester to conceptualize the components that might be required to run the bell. Now with more information on the transformer and the setup used to connect the bell to the transformer a detailed and specific design was implemented. Given below in Fig 38 is the schematic used to activate the chime.

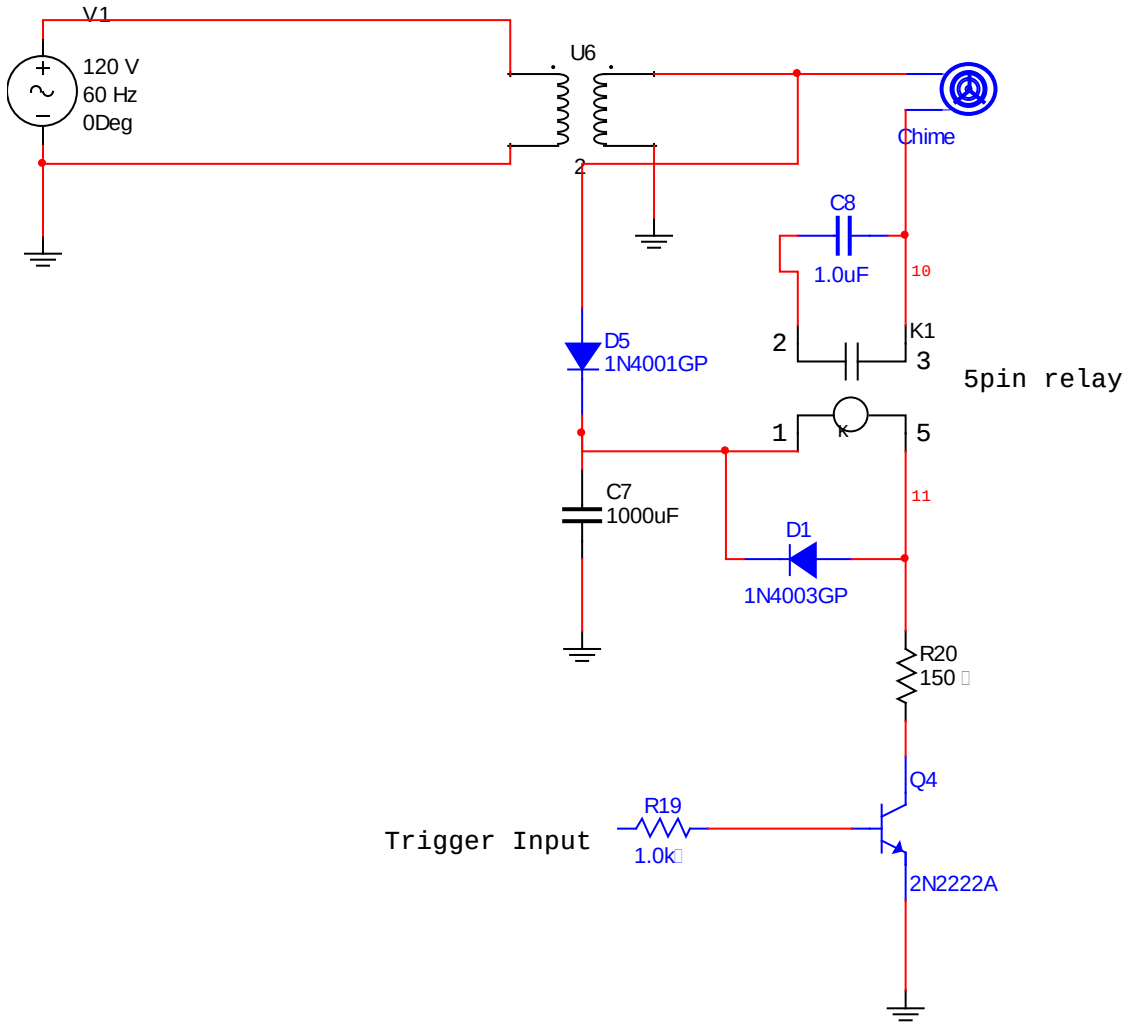


Figure 38: Bell activation circuit

The capacitor C7 is used an energy storage and maintains the voltage at a constant 25V. The diode D5 allows current to flow in the forward direction as pointed by the diode. A relay switch is used that switches automatically depending on whether current flows through the inductor (pins 1 and 5 of the switch). When pin 3 of the RS232 goes high, it sends a short pulse that is eventually received by the transistor Q4. This triggers the transistor to start conducting and thus current begins to flow from the transformer and through the switch. This causes the normally open switch to close and create a connection between pin 2 and 3 instead of pin 2 and 4. This completes the circuit creating a ground connection to the door chime. This in turn allows current to flow through the encasing, retracting the solenoid. When the pulse stops flowing through the transistor, the current stops flowing, and the solenoid is released quickly, hitting the metal strips, creating a chime.

Component Modifications

Section 2.8: Component Modifications
Distance Sensor

Previously it was assumed that another distance measurement sensor would be used to determine the baby's location once the robot had found the baby. This would be useful while following the baby around the room. But after attaching the Sharp GP2D02 sensor onto the radio turret of the robot no signal was being received. As per the robot manual and the distance sensor manual the connections were made as required. Yet when a signal was sent from the robot no information was received on the digital port or the bus terminals. It was not determined whether additional programming of the robot's processor was necessary in order to send the clock signal to the distance sensor. Possible reasons for not receiving a signal could be incompatibility between the sensor and the robot, or a faulty sensor.

Also it was noted that once the sensor was attached to the front of the robot, the robot would not be able to locate the baby if the baby was off to the right or left side of the robot. Due to these limitations the Sharp GP2D02 sensor and necessary connections to the robot were removed.

Instead the robots 8 infrared (IR) sensors were used to keep track of the baby. This design was less complicated since these sensors were already integrated into the base of the robot. Also since they all faced different directions it was easier to note the presence of objects that might be in the general vicinity of the robots perimeter. One limitation is that these sensors are mainly for obstacle avoidance and thus are more accurate to detecting the presence of obstacles that are very close to the robot, around 1-3 cm away. Thus keeping this in mind the distance between the robot and the baby was decreased from 11cm, as reported last semester, to 5cm. This ensured that the robot will be able to detect the presence of the baby while following it around the room.

General I/O Turret

During this project the General I/O turret was connected to the robot but unused due to the omission of the Sharp GP2D02 distance measurement sensor. However, the turret was not removed so that the height of the robot would remain constant at 10cm. The main reason for this is that during the testing stage the robot's height was 10cm. If the height of the robot is changed at this point it would cause the field of view of the camera to change. This would cause an unnecessary repetition of the tests. Assembling and disassembling of additional turrets is a delicate operation. It should be avoided as much as possible and performed carefully. Improper assembling or disassembling could damage or bend the pins on the turret.

Software Modifications

Section 2.9: Software Modifications
Path Planning Algorithm

The research and design of the path planning algorithms was an integral part of the conceptual design generation last semester. This semester however it was not an integral part of the design process since the robot always took the shortest path to the baby. This was easily accomplished with the help of the search path that the robot uses in the beginning of the process to find the baby. By reducing the distance traveled, from 35cm to 25cm, before the robot takes the picture it was possible to decrease the number of obstacles between the robot and the baby. Hence, when the robot finds the baby it is usually behind only one obstacle and possibly near another one. This allows the robot to travel around the obstacle and reach the baby quickly since it then travels in a straight line.

LabVIEW Processing

As mentioned earlier, LabVIEW processing was to be used to obtain information from pictures taken by the robot. This process was to allow the user the ability to determine distances using the Vision based Depth Perception process. Unfortunately, after numerous trials it was determined that to acquire an image for processing, the camera turret needs to be equipped with a National Instrument's processor. Since this was not the case with this robot the image acquisition method was not possible. Also, since the basis or the project was a fully autonomous robot that had the capabilities to process information without external help the LabVIEW processing method would not work. This process requires modifications on the image that are to be run by the user. Instead, now the robot processes the information autonomously using C programming.

Finding the Baby

Previously it was assumed that the camera turret has the capabilities to detect certain colors and activate flags when required. But due to the poor quality of the camera it was determined that this process to find the baby would not work since the robot cannot determine whether a pixel is yellow or not based on the raw image data. The raw image data is simply an array of numbers that are generated by the camera processor. A filtering process had to be designed that would use this information to determine whether there exists yellow pixels, that represent the baby, in the image. Now instead of activating a flag the robot turns on an LED, which is present on the base, to indicate that the baby is found. A photo of the robot with this LED on is shown below in Fig. 39.

Figure 39: Baby Found LED

SECTION III

Testing and Results

Measured Parameters

Section 3.1: Measured Parameters

This section describes the parameters and quantities that will be needed in the testing stage of the project. Also, it will describe the processes required to indicate the accomplishment of each task. These indications are for users to realize that the robot has completed a task and it will be proceeding to the next stage.

Lego Obstacles

Once the environment has been set up as described in Section 1.2 the obstacles will be placed around the room. The dimensions of a 4X4 Lego block are 16x16x9.6mm. Dimensions of the Lego blocks will be varied by shape and size during the testing process. The height of the obstacles will not exceed 42mm.

Time Considerations

Finding the baby: To find the baby the robot will have 5 minutes to complete this task. If the baby is found then the finding process will be complete and the robot will move on to the next stage. In case the baby is not found the robot will set of the alarm. A stopwatch will be used to keep track of the time. An internal timing method will have to be determined to notify the robot when 5 minutes have elapsed.

Serial Cable: The length of the S serial cable is limited to two meters for proper operation. It will take about 2.667 seconds for an image to be transferred from the robot to the computer or vice versa. Due to this model being a prototype system, operation will not be taking place in real-time. Thus, during image transfer and processing the robot and the baby will be rendered idle. Once the robot obtains the necessary information from the data or image processing the test will resume. Also, a method will be devised to calculate the actual transfer time of the image during testing.

Scenarios

Different scenarios will be required to test the various tasks of the robot. There will be three levels of complexity for each test in order to determine how well the robot will perform even in simple as well as complex environments. This will give an estimate of the limit of the robot in diverse environments. The general location of the obstacles, the robot and the baby has been depicted in the respective scenario figures.

Scenario I: This will be a simple case where the robot can easily find the baby as shown below in Fig. 40. This means that the baby will be initially in the line of sight of the robot so that the robot can instantly find the child. There will be no obstacles placed around the room so that the robot can navigate around the room with ease.

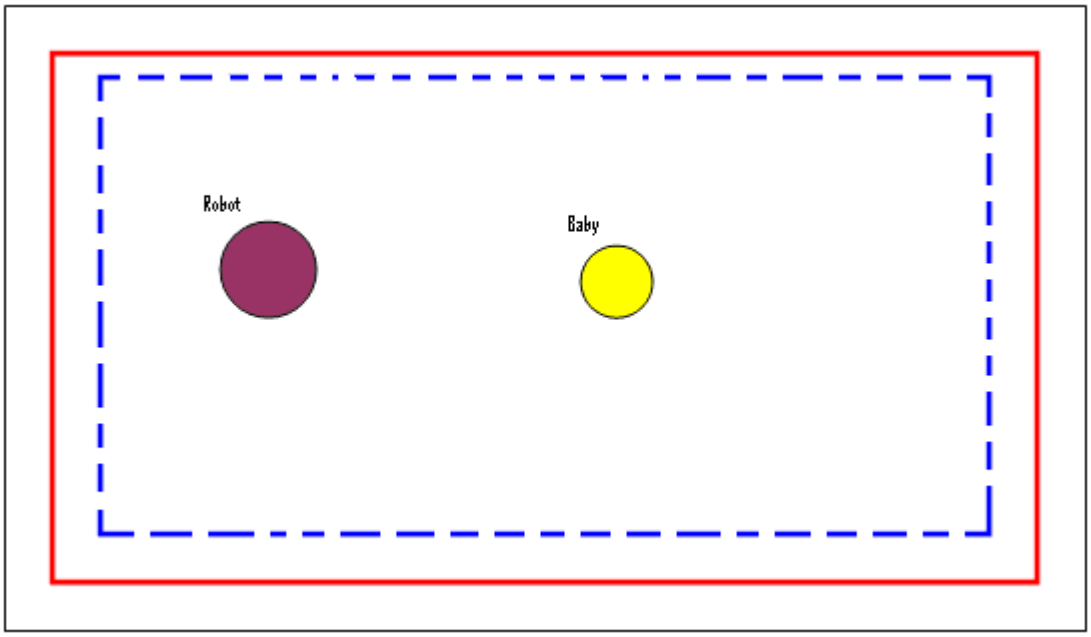


Figure 40: Top view of scenario I

Scenario II: This setting will have an intermediate difficulty level and is depicted below in Fig. 41. Thus, even after the robot has located the child there will be obstacles in the path that the robot will have to avoid in order to get to the child. During this scenario a single obstacle will be placed between the robot and the child.

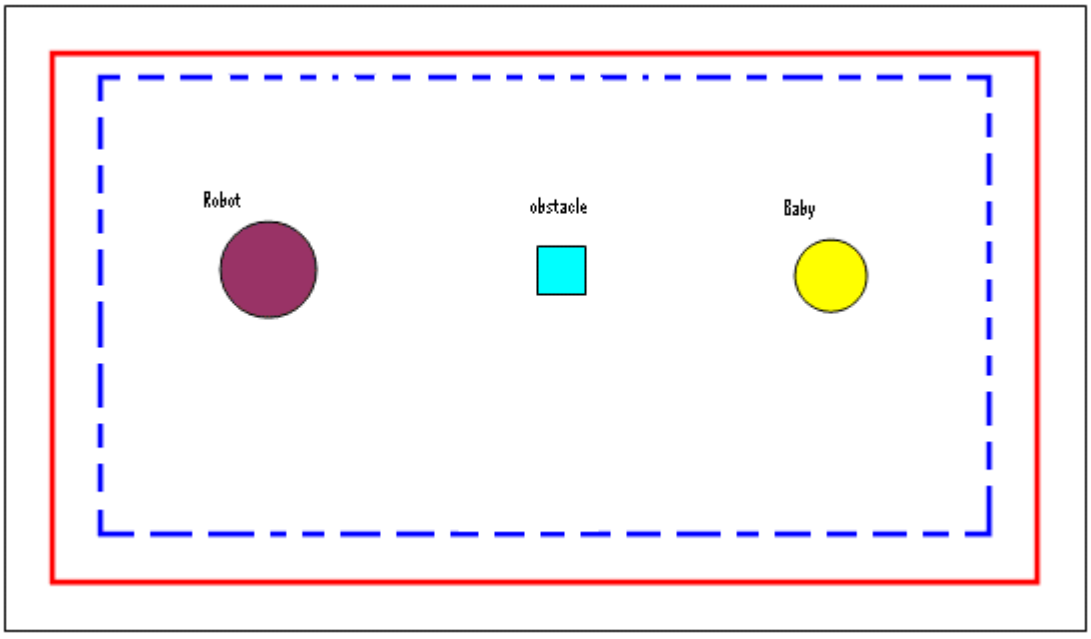


Figure 41: Top view of scenario II

Scenario III: This will be the most difficult situation for the robot. The obstacles will be placed in such a way that the robot will have to scrutinize the pictures carefully to find the child. Then the obstacles will be placed so that a complex solution is determined to reach the baby. Also the structure of the obstacles will be more composite as compared to the previous obstacle and is shown below in Fig.42.

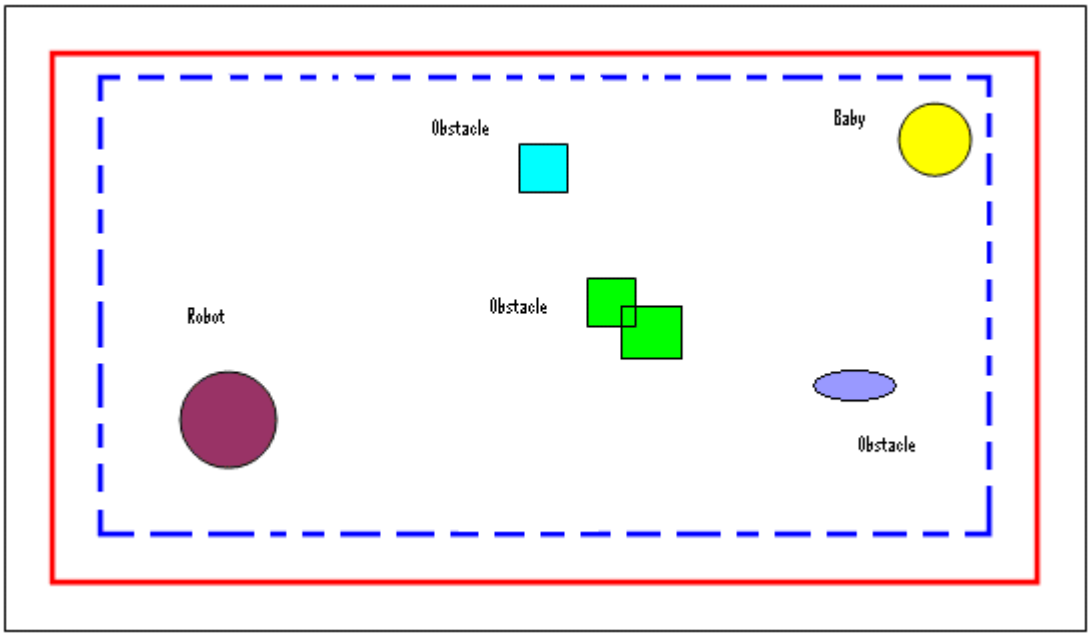


Figure 42: Top view of scenario III

LED Indication

After the task of finding the baby the robot will turn on an LED to indicate the completion of the task. This will also indicate that the robot has moved on to the next stage of the process. A mechanism would have to be devised to turn on the LED's as required, either by the robot or by the computer.

Accuracy

If the robot succeeds 8 out of 10 times the test is run it will indicate a success for the total scenario. Thus it is assumed that if the robot is successful 80% of the time then the robot has successfully accomplished the task.

Table of Results

Each time a task in a particular scenario is tested the results will be categorized in tabular form depending on the outcome of the test. Given below in Table XX is a generic table generated for the finding stage.

	N	Y
N	Num1	Num2
Y	Num3	Num4

Table XX: Result of testing

The horizontal direction represents for the ground truth. The vertical direction represents for the machine response. The number of successful runs will be the sum of Num1 and Num4; while the total number of unsuccessful runs is the sum of Num2 and Num3. Each of the numbers will be a total sum of the number of times each task was tested for each scenario.

For example, if the test to find the baby is tried 10 times. 5 times when the baby is actually present (ground truth = True) in the room and 5 times when the baby is absent (ground truth = False). Then some sample results could be that the robot has found triggered the alarm (after 5 minutes) 4 times when it did not find the baby and once it actually finds the baby when the baby is not present. Also, the robot successfully finds the baby 3 times and fails to find the baby twice when the child is present. The results are displayed in Table 2 below.

	N	Y
N	4	2
Y	1	3

Table 2: Result of Finding Task

Then the success rate would be = (4 + 3)/10 = 7/10 = 70%.
This would be assumed as a failure by the standards of this testing process.

Consistency

To determine how consistent the robot is in each environment the scenarios will be tested numerous times without any changes to the environment. Based on a scale from 0-10 a percentage rating will be assigned after each test. This could indicate any glitches that might be present in the robot or the software.

Variability

The obstacles will be moved to create different environments but with the same complexity level. Further, the robot's starting position and the baby's starting position might be changed as well. This will determine how the robot will react in different environments. This will be more analogous to real environments where the robot might be placed in changeable environments.

Hardware Design

Section 3.2: Introduction

One of the main purposes of the hardware design was to generate a flashing lights circuit to distract the child from going into dangerous situations. Four bright LEDs (light emitting diodes) were placed in the four corners of the room. This guarantees that the baby will see at least one light no matter which direction it is facing.

Next, an alarm system was necessary to alert the parents for three different purposes. First if BabyBot doesn't find the baby after searching the entire room, the robot needs to alert the parents. Next, if the robot is trapped between the wall and the baby, i.e. the baby might pick the robot up and BabyBot has no room to back up, it needs to send and alarm to alert the parents. Finally if the baby is in the out of bounds area the robot needs to notify the parents that the child might be in danger.

To implement these two aspects of the design the radio capabilities of the robot are used. When necessary the robot will send a wireless signal to the radio base to activate the necessary circuit. The RS232 connection on the radio base is connected to the breadboard using a standard ribbon cable with 10 wires. Given below in Fig 43 is the sample of the pulse that will be sent from the RS232 cable when a signal is sent, from the robot, to the radio base.

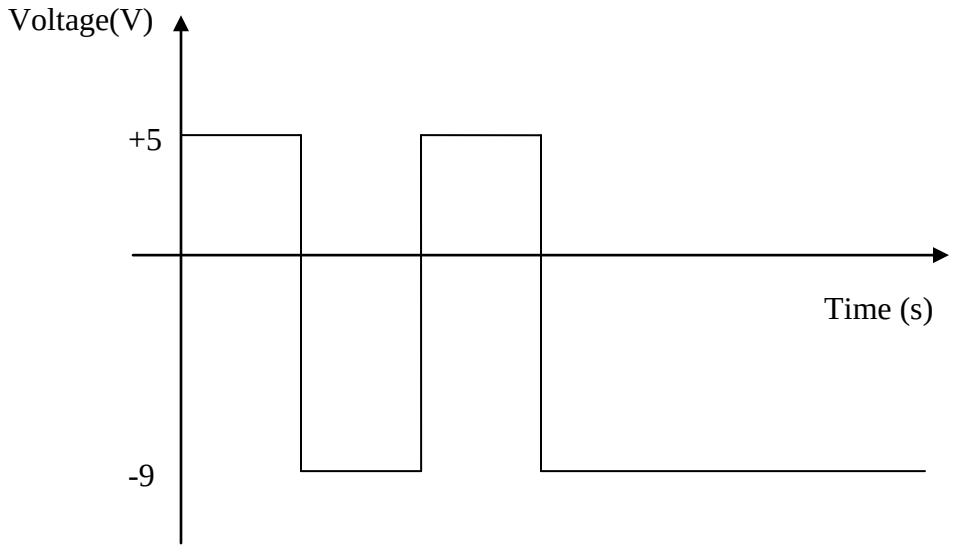


Figure 43: Sample signal from radio base

A complete circuit diagram is shown below in Fig 44.

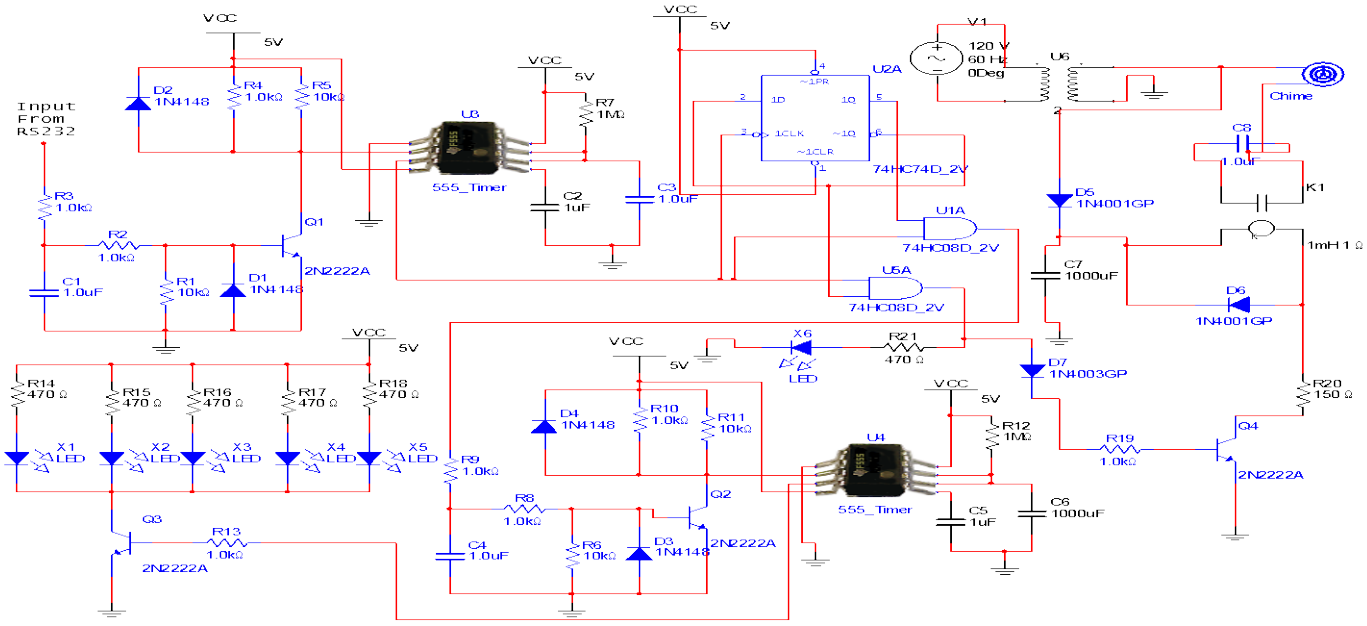


Figure44: Complete circuit

LED CIRCIUT

The circuit given below in Fig. 45 is the LED portion of the circuit. As previously stated these LEDs are used to distract the child. Further more this distraction is intended to keep the child from entering into the out of bounds area. The changes in this circuit include the placement of 4 of these LEDs in the corners of the testing table instead of on the breadboard. Also, these 4 LEDs are of the super bright type with a rating of approximately 5000 mcandelas.

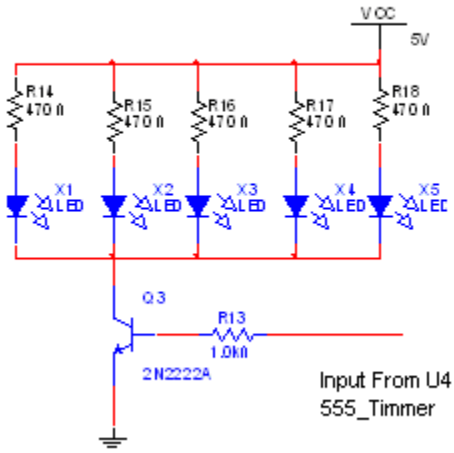


Figure 45: LED Circuit

TIMER CIRCUIT

The circuit shown below in Fig. 46 is used to increase the input pulse from 10ms to a longer output that can be used to drive different aspects of the hardware output. Unlike other 555 timer circuits, this circuit is triggered on the rising edge of the voltage instead of the falling edge. The input (pin 2) is set high and once a trigger is detected the transistor is turned on so that the output is triggered.

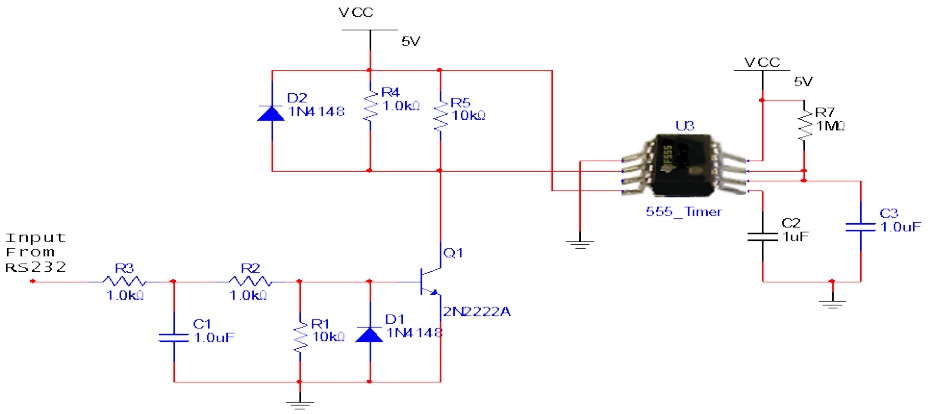


Figure 46: 555 Timer circuit to extend pulse to 2s

Changing the values of the resistor R7 and capacitor C3 will affect the amount of time the output stays high. Thus for the LED circuit the output is required to stay on longer so that the baby has a chance to see the LEDs. Thus the capacitor and resistor values were increased to keep the output high for at least a few more seconds, as shown below in Fig 47.

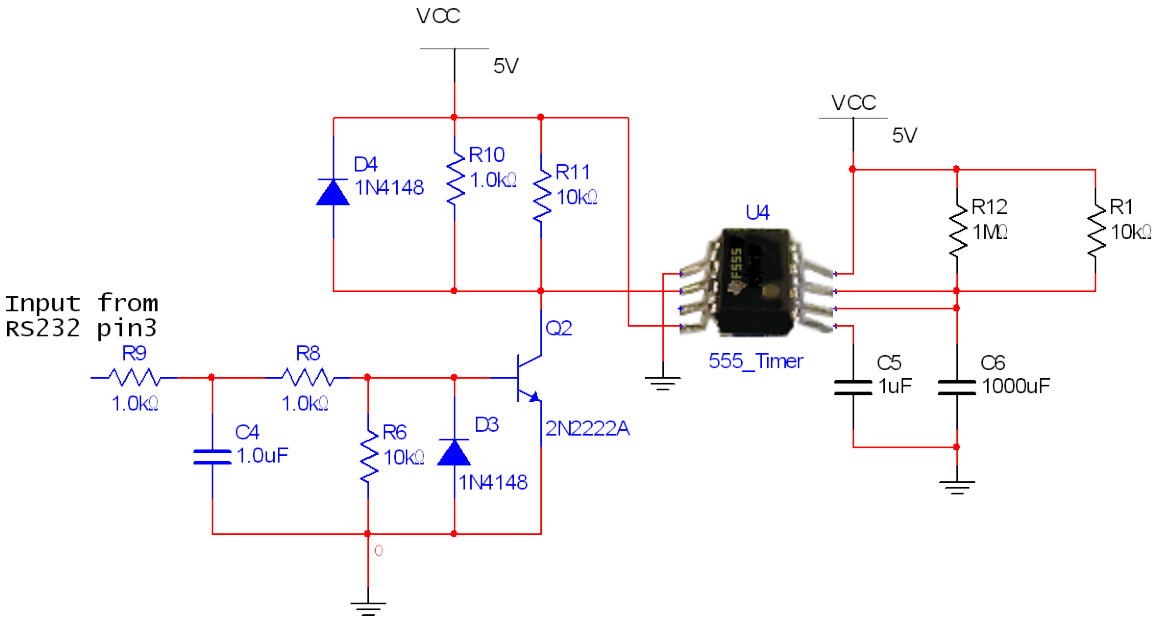


Figure 47: LED trigger circuit

Alternating Between the Two Circuits

The output of the timer goes to the 7474 D- flip-flop. This serves the purpose of alternating between the LED circuit, and the bell circuit and is shown below in Fig. 48. The $\overline{Q}$ output of the flip-flop is reversed back to the D input and the timer output acts as the clock. The Q, $\overline{Q}$ outputs are ANDed with the clock. This ensures the only one circuit is ON at any particular time. The set and reset pins are pulled high for proper operation.

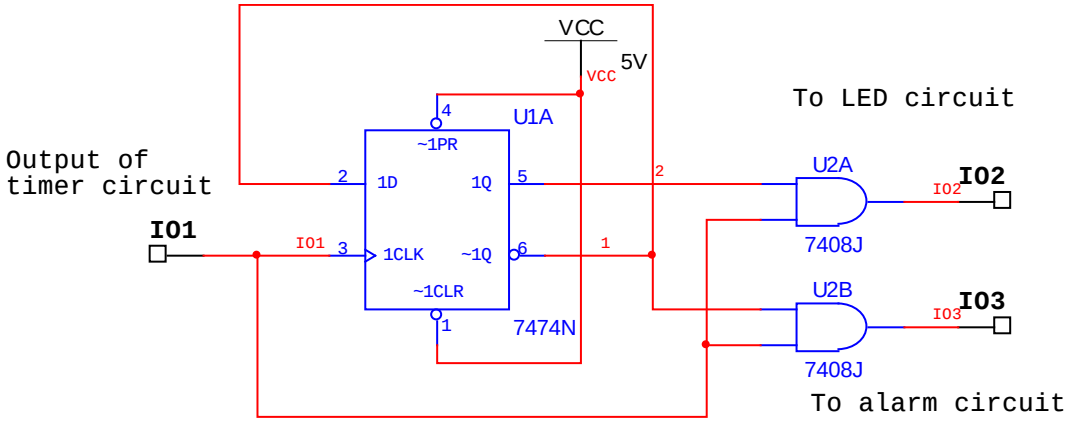


Figure 48: Alternating circuit design

RS232 Signal to the Breadboard

When a signal is sent from the radio turret of the robot to the radio base, the Carrier detect (CD) and the Receive radio data (data transfer from the radio turret to the base station, RXHF) indicator LEDs come on confirming that a signal has been received by the radio base. Immediately, the Transmit RS232 data (data transfer from the base station to the breadboard, TX) LED comes on for a fraction of a second indicating that a signal is being sent from the radio base to the bread board through the RS232 connection. To connect the radio base to the breadboard first a converter was used to convert the DB25 connection down to a DB9 connection as shown below in Fig. 49, along with the schematic pin out.

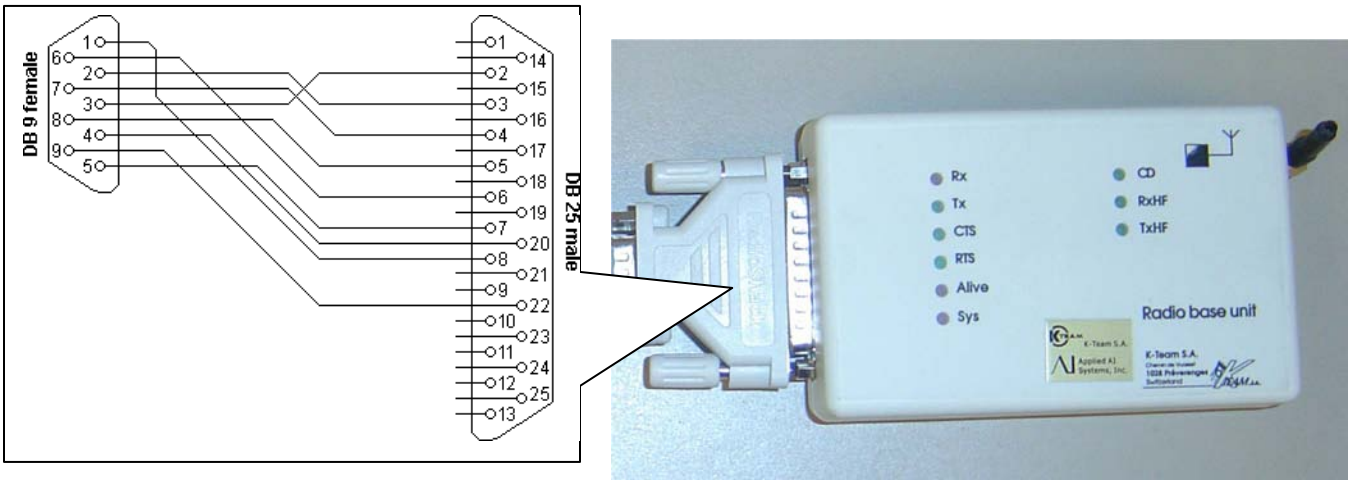


Figure 49: Radio base and converter

Then a ribbon cable connector (with 10 wires) was used to connect the DB9 connector to the breadboard. Pin3 (signal) was connected to the input of the first timer circuit. Also, pin 7 was connected to the ground of the breadboard circuit. The ribbon connector is shown below in Fig. 50.

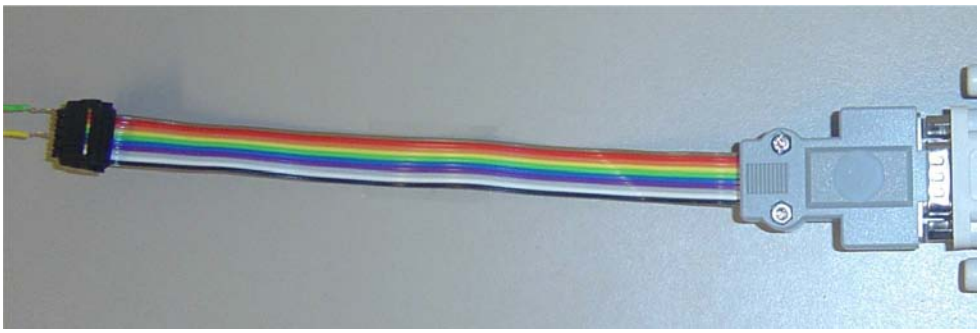


Figure 50: Ribbon connector

EXPERIMENTAL RESULTS

Date: March 23, 2006

Pulse Time Estimation

The time that the output is turned on for depends on the resistor (across pin7-V_{CC}) and the capacitor (across pin8 –GND). Namely,

$t = 1.1 \times R \times C$

Pulse Time Estimation

Circuit	Resistor Value(Ω)	Capacitor(μF)	Experiment Value (s)	Calculated Values(s)
Bell	1.1M	1000	11	12.1
LED Circuit	1M	1	2.5	1.1

Output Values for Timer Circuit

While measuring the voltages the output was not connected to any other circuit.

Timer Circuit Output

Pin Number	ON Voltage(V)	OFF Voltage(V)
1	0.00	0.00
2	1.19	4.99
3	3.40	0.00
4	4.98	5.00
5	3.28	3.30
6	0.00	0.00
7	0.00	0.00
8	5.00	5.00

Alternating Circuit

Instead of using the bell an LED was used to determine whether the voltage changes as required. $\overline{Q}$ and Q change alternatively depending on the clock. For example, if $\overline{Q}$ is 4.41V it will hold that value until Q changes, then $\overline{Q}$ will go back to its original value. The LED is currently used as an indicator to whether the bell is receiving a signal. It should be noted that when Q is on $\overline{Q}$ will be off. But their on/off voltages are listed below together.

Alternating Circuit Values

Signal Name	ON Voltage(V)	OFF Voltage(V)
Clock	4.98	0.01
$\overline{Q}$	4.41	0.09
Q	4.40	.13
Base	0.76	0.05
Emitter	1.19	4.99
Bell input	3.18	0.05

RS232 Signals

Pin Number	ON Voltage	OFF Voltage
3	N/A	-9.20
7	0.00	0.00

Section 3.3: Robotic Component Evaluation
CAMERA

Image Processing

The raw data obtained from the image matrix was used to determine the type of colors and quality of the pictures taken by the robot. The processor on the camera turret is a VV6300 manufactured by VLSI Vision Limited. It is an integrated CMOS (complementary metal oxide semiconductor) color image sensor with an on-chip ADC (analog/digital converter). The standard image format is a 120X160 (row x column) matrix. The raw data for these pixels is stored in the Bayer pattern color pixel array.

In this pattern each pixel is divided up into its respective red, green and blue values that can range from 0 to 255. The higher the value the more concentrated the color is in that pixel. Given below in Fig. 51 is the matrix along with the row and column numbering.

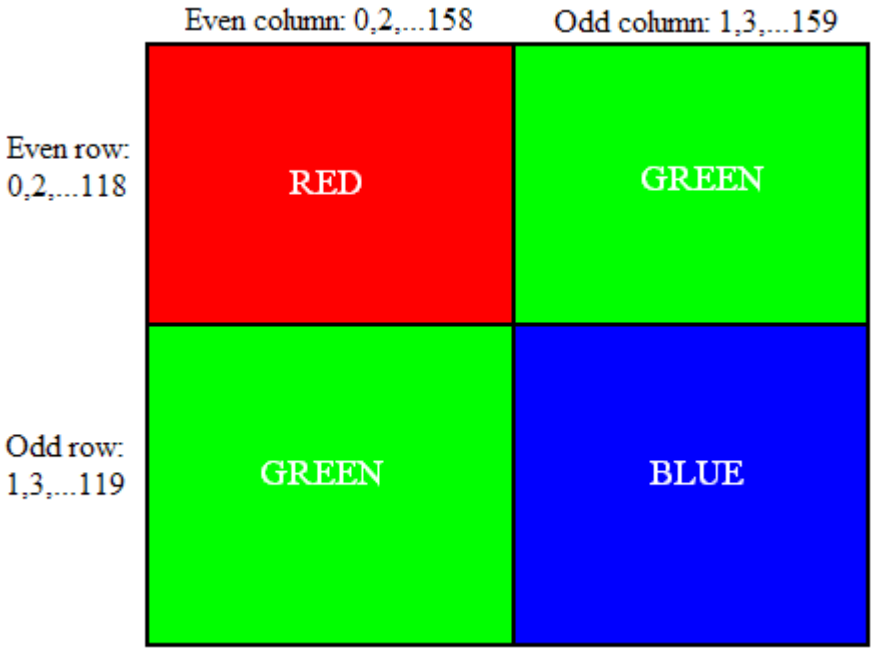


Figure 51: Bayer filter pattern

Using this data it was possible to extrapolate a section of the image where an object of a particular color was placed. Then an average value of the surrounding pixels was used to determine what the robot was looking at.

METHOD 1

During the first trial the blocks were placed farther from the robot (about 12cm) and the lower left corner of the matrix was used to obtain the results. On the next trial the object was placed closer to the robot (about 2-5cm) and the top left corner of the matrix was used for extrapolation. An average value of the Red, Green and the Blue content of the pixel was obtained. When these average values were put into a graphics program, such as Paint, the results were rather unexpected and different from the actual color of the object. Fig. 52 illustrates the process of obtaining the result once the average values for the three colors have been obtained. Next the results of Trial I follow, along with the image comparison.

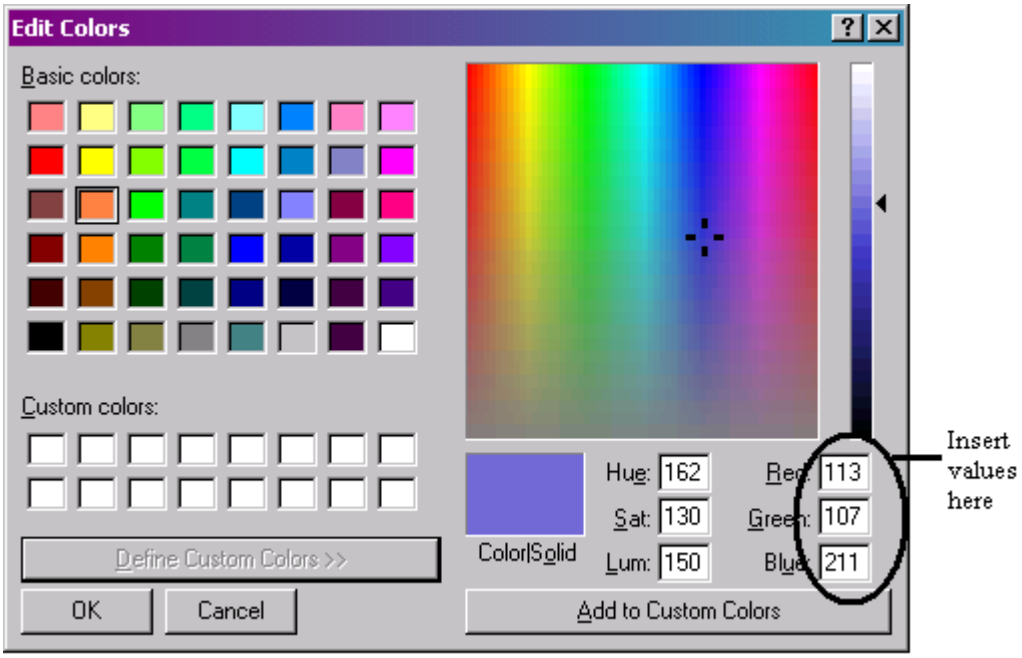


Figure 52: Verification of color after extrapolation process

Date: April 5, 2006
Blue Lego Block Trial

Row/Column	0	1	2	3
116	66	91	74	95
117	87	55	95	63
118	67	89	74	100
119	89	57	99	63

$$Red = \frac{66 + 74 + 67 + 74}{4} = 70.25 \approx 70$$

$$Green = \frac{91 + 95 + 87 + 95 + 89 + 100 + 89 + 99}{8} = 93.125 \approx 93$$

$$Blue = \frac{55 + 63 + 57 + 63}{4} = 59.5 \approx 60$$

A B

Figure 53: A) Result from Paint, B) Image from Camera of Blue Block

Red Lego Block Trial

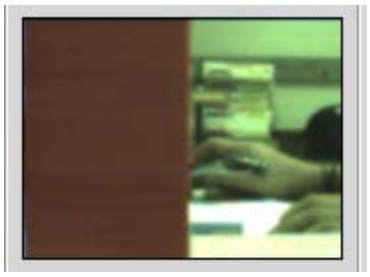
Row/Column	0	1	2	3
116	77	49	71	45
117	47	35	49	37
118	76	52	77	51
119	50	32	49	35

$$Red = \frac{77 + 71 + 76 + 77}{4} = 75.25 \approx 75$$

$$Green = \frac{49 + 45 + 47 + 49 + 52 + 51 + 50 + 49}{8} = 49$$

$$Blue = \frac{35 + 37 + 32 + 35}{4} = 34.75 \approx 35$$

A



B

Figure 54: A) Result from Paint, B) Image from Camera of Red Block

Yellow Lego Block Trial

Row/Column	0	1	2	3
116	96	110	98	116
117	111	57	116	59
118	97	113	96	112
119	115	55	116	58

$$Red = \frac{96 + 98 + 97 + 96}{4} = 96.75 \approx 97$$

$$Green = \frac{110 + 116 + 111 + 116 + 113 + 112 + 115 + 116}{8} = 113.635 \approx 114$$

$$Blue = \frac{57 + 59 + 55 + 58}{4} = 58$$

A B

Figure 55: A) Result from Paint, B) Image from Camera of Yellow Block

Estimation of Camera's Field of View

This test is to detect the limitations of the field of view (FOV) of the camera since there is no information about this in the K6300 camera turret manual. The baby was placed 10cm away from the robot and each time the baby was moved to different position. The angle between the front of the robot and the baby was varied. Given below in Fig. 56 are the positions of the baby (numbers 1-6) relative to the robot. The results of the experiment follow. If the baby was found, the row and the column where the baby was found are also shown. But, if the baby was not found then the numbers are not-applicable (N/A).

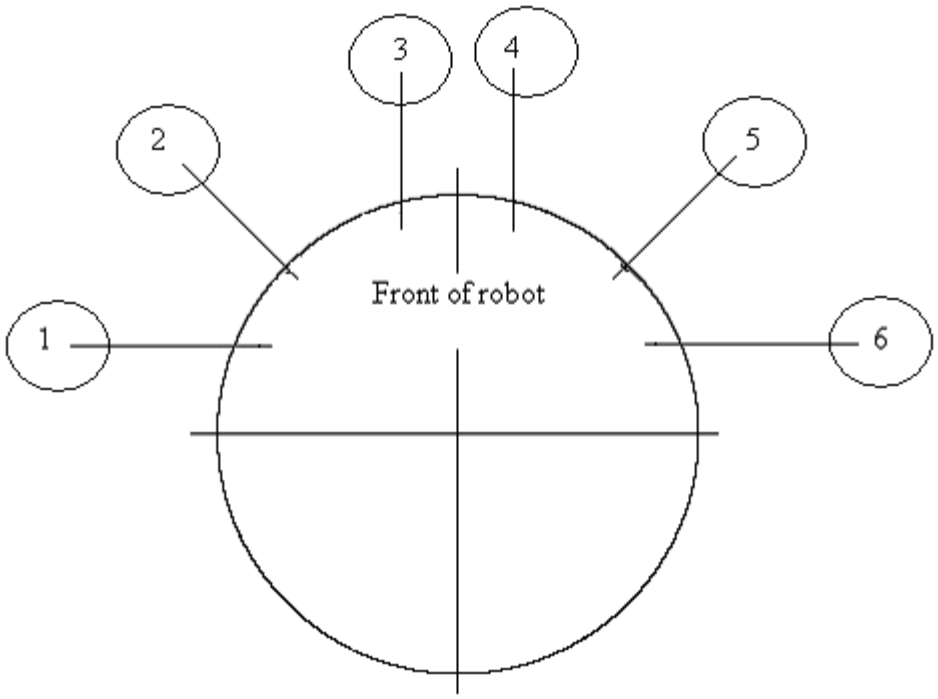


Fig 56: Position of Baby relative to Robot

Tabular Result – Camera FOV

Position	Row	Column
1	N/A	N/A
2	N/A	N/A
3	3	64
4	1	100
5	N/A	N/A
6	N/A	N/A

It is estimated that the angle of view of the camera is approximately 30^0.

Analysis

It is noticed from the above experiments that the pixel values obtained do not necessary correspond to exactly the real colors of the objects itself. The raw data has a lot of variations as compared to the actual colors as seen by the human eye. Thus it was concluded that color ranges for the pixels cannot be obtained from a graphics program, such as Paint. These results would not correspond accordingly. For example, if the robot needs to detect the yellow pixel it is not possible to do so by placing the range for the red, yellow, and blue as one would obtain from Paint. One reason for this is that the quality of the image is very low, thus the colors are distorted as compared to their actual values. Hence it was concluded that alternative methods are required to filter the image to detect particular colors.

It should also be noted that the processor on the camera turret compensates for any changes in the quality of light. For example, if a picture is taken in a slightly darker room, the contrasts between the colors will be more enhanced, although the colors themselves might be more diluted. Also due to the low quality of the camera, the farther the objects the darker or hazy the images gets.

The field of view of the camera is not very large. Only objects that are directly in front of the camera are detected by the robot. The objects distance away from the robot has no affect on the field of view. That is if the object is on the right hand side (position 5 or 6) of the robot then it doesn’t matter whether the object is 10cm or 40cm away, the robot will not see it.

METHOD 2

Due to the low quality of the images received from the camera this alternative method was developed. This method allows the appropriate filter ranges to be determined so that the objects are distinguishable even though the colors are slightly distorted.

An object was placed directly in front of the robot so that the image obtained would contain only the color of the object and the white background. Next, an image was obtained using the robot's camera turret. After the image was obtained the matrix values corresponding to lines 60 and 61 of the image were read. The matrix values represent the levels of red, green and blue in each pixel. These numbers were then transferred into a spreadsheet program that displayed the results in a graphical form. From the graph related to each subject the red, green and blue filtering levels were determined.

Yellow Lego blocks were used to represent the child for this project. As a result the yellow blocks were the first to be assessed. Given below in Fig. 57 are the results of the yellow blocks matrix.

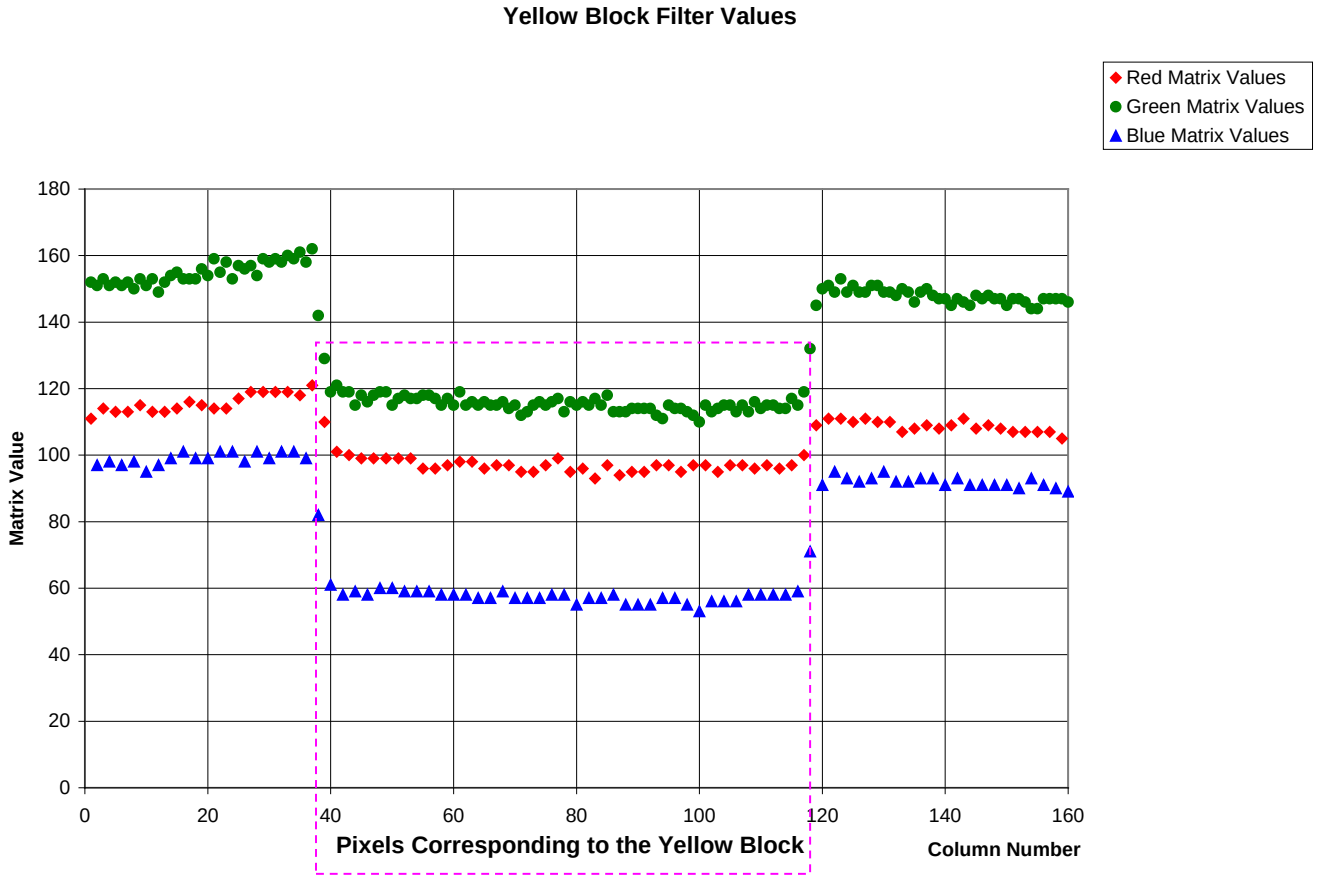


Figure 57: Yellow block filter values

There will be obstacles present during some of the different scenarios that are tested. Subsequently the purple objects were also tested for filter values as shown below in Fig. 58.

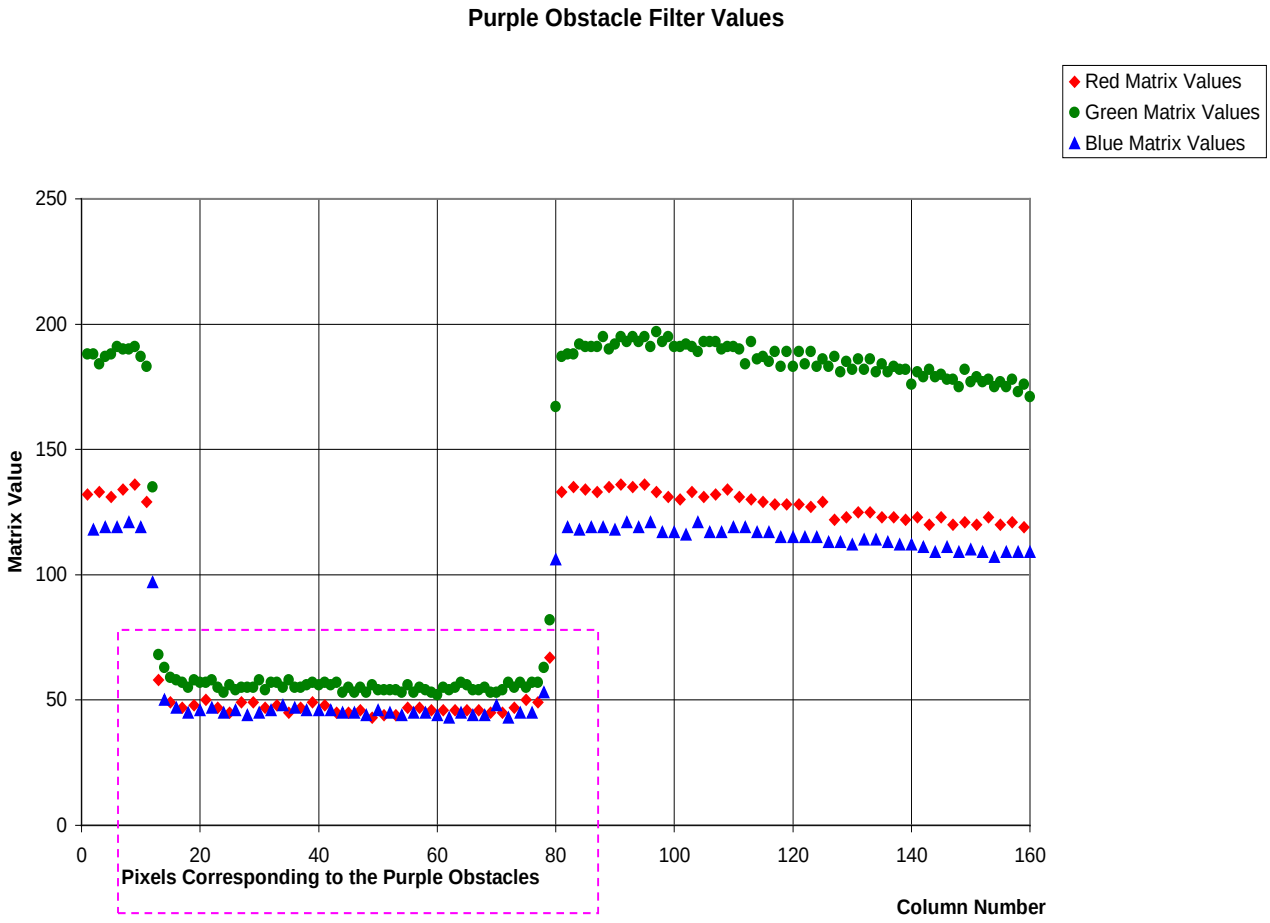


Figure 58: Purple obstacle filter values

The red out of bounds marking on the table was also taken into consideration for this test. The results are shown below in Fig. 59.

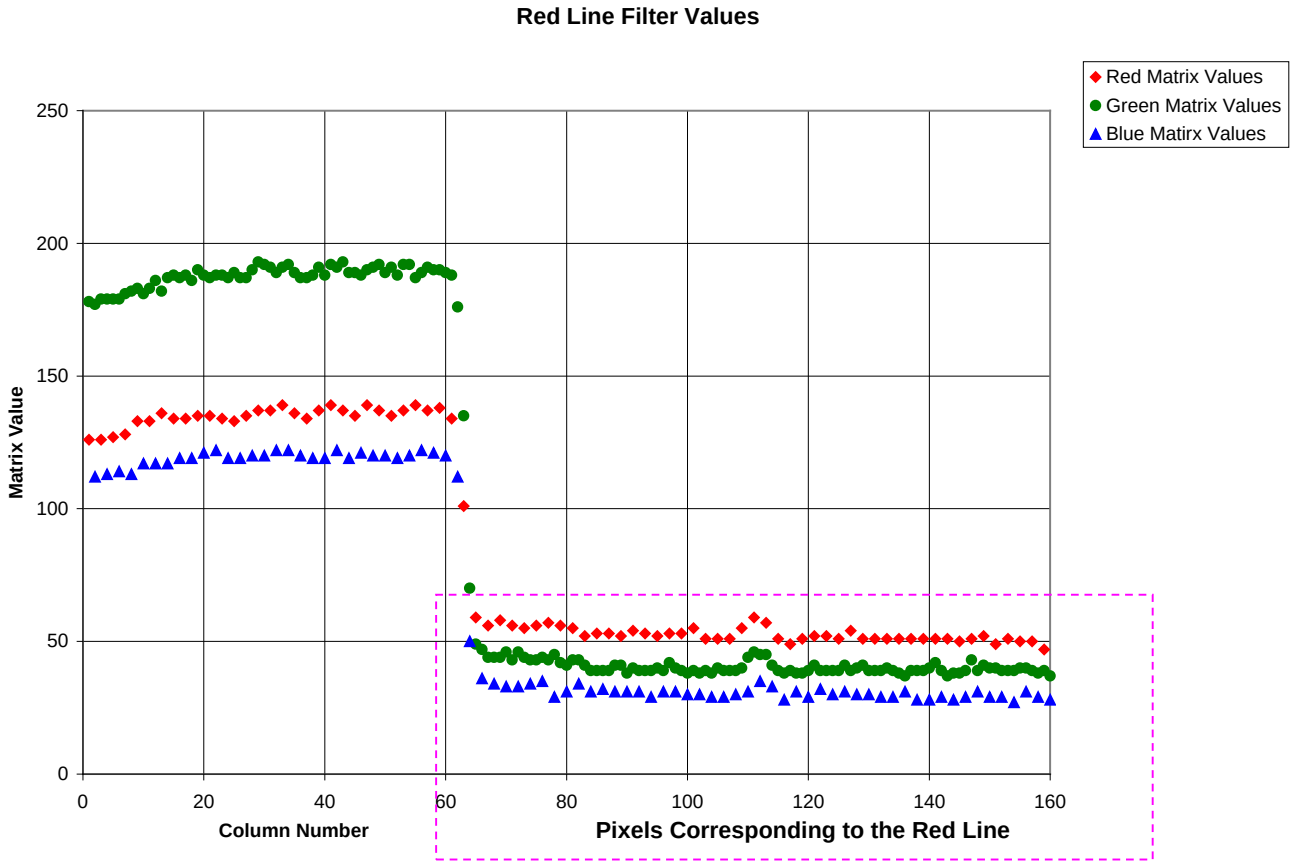


Figure 59: Red line filter values

Finally, the last object to be tested with this method was the blue warning line from the table. The red, green and blue matrix values are shown below in Fig. 60

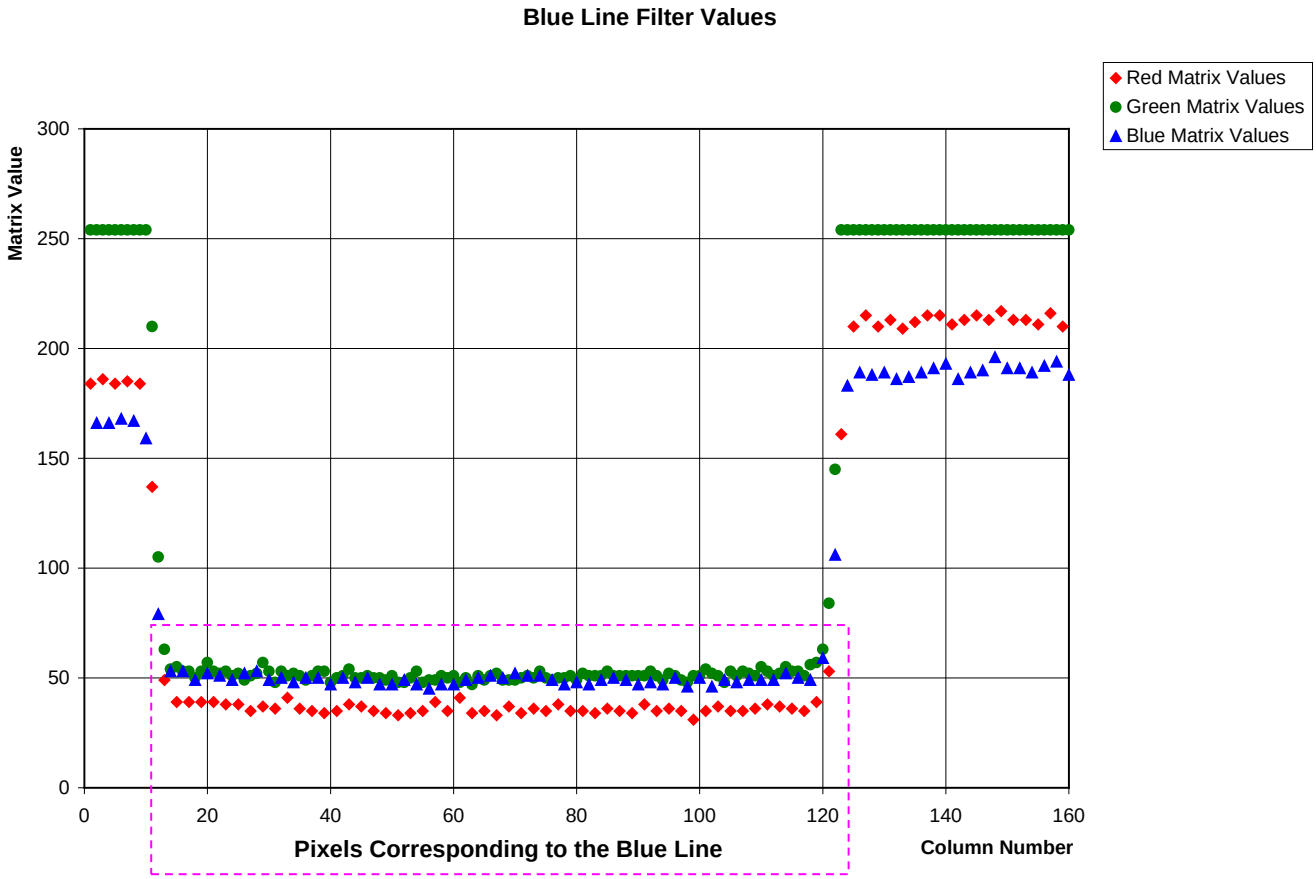


Figure 60: Blue line filter values

Therefore, the experimentally obtained red, green and blue filter values are given in the table below. These are the approximate extreme limits of the matrix values obtained from the graphs.

Filter Values

	Experimentally Obtained Filter Values					
	Lower Red	Upper Red	Lower Blue	Upper Blue	Lower Green	Upper Green
Yellow Blocks	95	100	50	60	110	150
Purple Obstacles	44	60	44	60	47	65
Red Line	47	59	28	35	38	46
Blue Line	31	41	45	51	48	57

Analysis

The use of this method allows for the filter values to be determined based on the way BabyBot differentiates colors. Even though these values may not correlate to the values that would normally be associated with each of the given colors, they provide accurate results since they are based on the way BabyBot sees the colors.

It takes the robot about a maximum of about two minutes to go through the complete matrix (120X160).

Conclusion

Thus, it was concluded that the range of values that the robot should look for, for each color, are accurately obtained by using this method. Also, now that these filter values have been obtained they can be used in the internal programming for the robot, to make the system more autonomous. This means the robot will not have to rely on the host computer for image processing. All computations are done onboard the robot's microprocessor.

Distance Estimation

It was noted that while taking the pictures the farther the object was from the robot, the higher it would be on the image. A sample of this is shown in Fig. 61 below. In the images shown below where the “baby” is moved away from the robot and images were obtained at different locations.

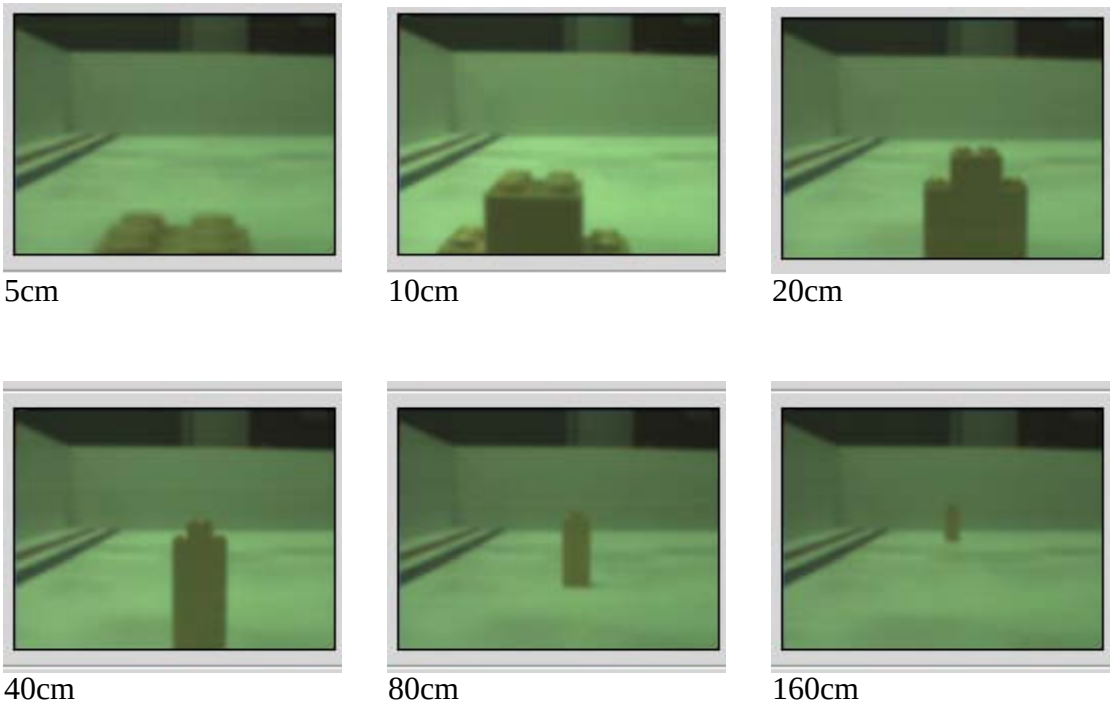


Figure 61: Distance images

Also, depending on which quadrant the baby is found in, the robot might be able to detect whether the baby is to its left, right or exactly in front of it. To detect if this was true images were taken and processed by the robot and an output was obtained as to the row and the columns when the robot detected the presence of the baby. The next section contains the detailed testing result.

Date: April 8, 2006

The distance between the robot and the baby were varied but the robot was always facing the baby head on.

Trial I

Distance	j(Row)	i(Column)
10	1	46
15	1	38
20	19	52
25	13	78
30	47	88
35	1	82
40	13	88

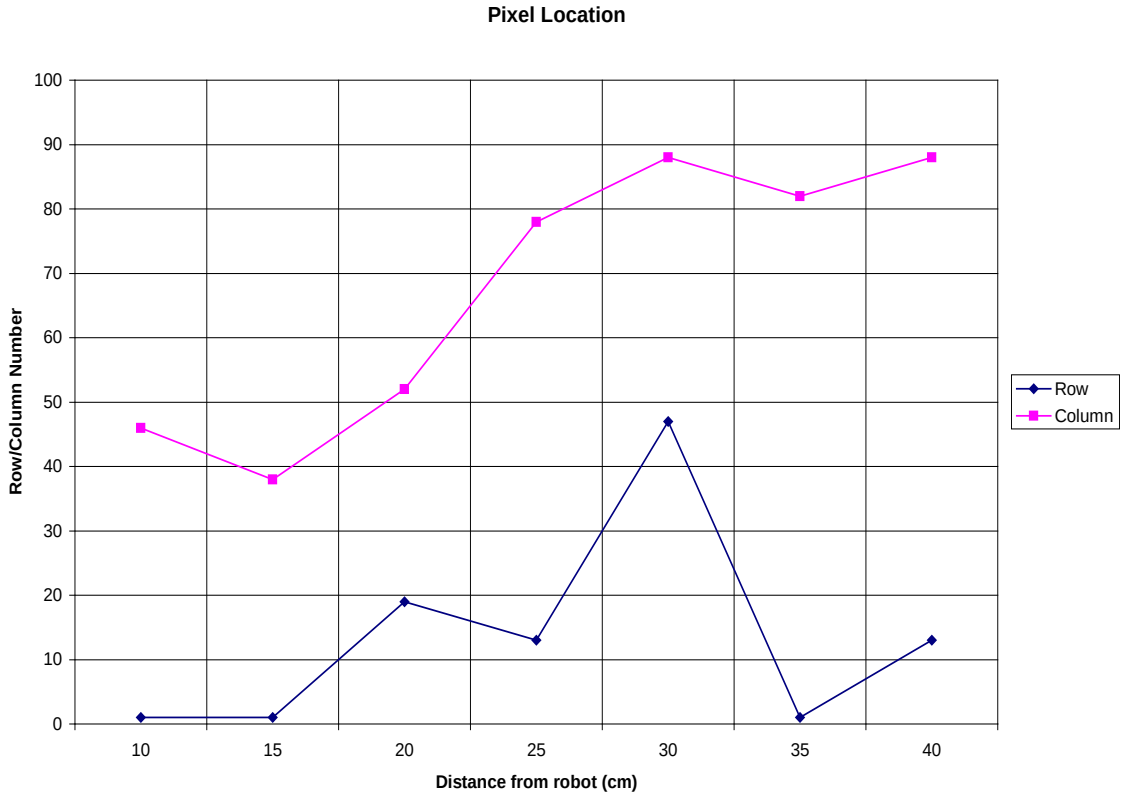


Figure 62: Trial I

Trial II

Distance	j(Row)	i(Column)
10	1	50
15	3	64
20	19	96
25	1	64
30	37	74
35	43	82
40	1	88

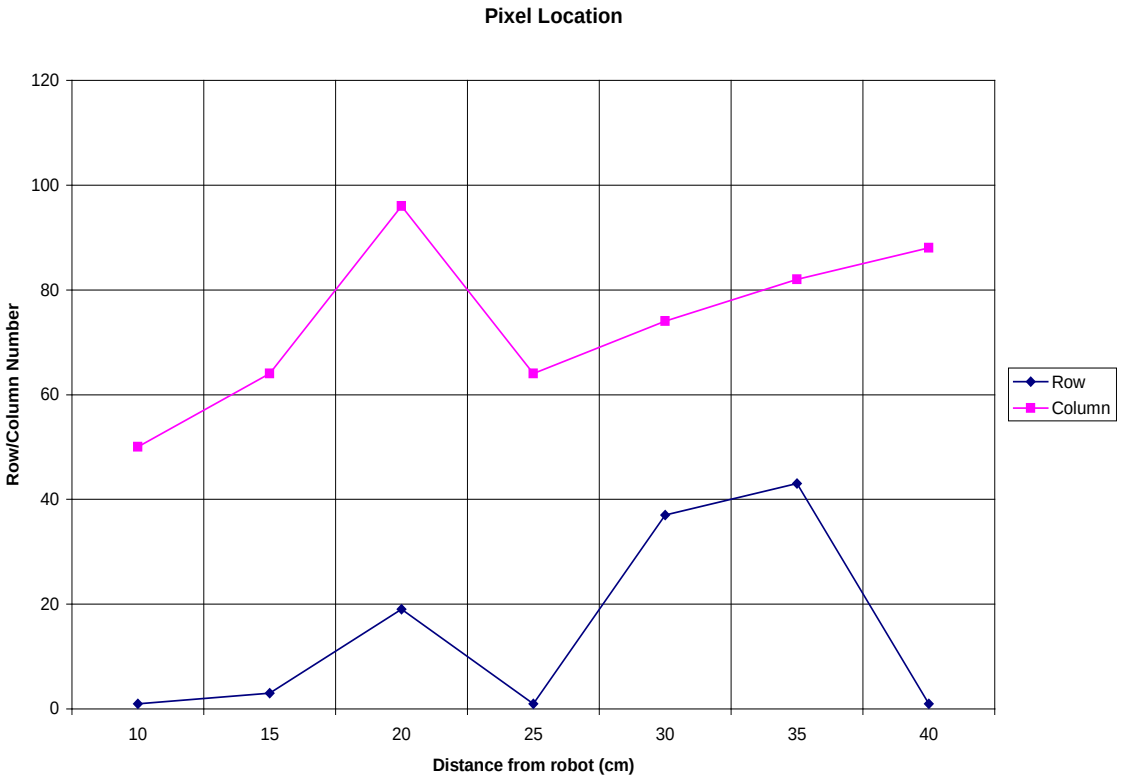


Figure 63: Trial II

Analysis

Due to the inconsistent nature of the output obtained it was concluded that this method would not produce accurate results that are required by this project. It is not possible to detect small changes in the distances between the robot and the baby. For example the result of row value was 1 even if the distance was changed from 10cm to15cm. Also as the object got farther from the robot, the column values increased accordingly. But, in reality these numbers should be consistently in the center of the matrix. Thus the camera image was used to detect objects present, but not used to calculate their distances from the robot.

Infrared Sensor Readings

The robot is equipped with 8 Infrared (IR) obstacle detection sensors, the TCRT1000 manufactured by Vishay Electronics. These sensors are used to detect obstacles that might be around the robot at any time during the experiment. They are also used to follow the baby around the room once the robot has reached the baby. To determine what types of values these sensors would provide, an object was placed at different distances away from the robot and the output reading for each of the eight sensors was measured. Given below in Fig. 64 is the position of the different sensors on the robot.

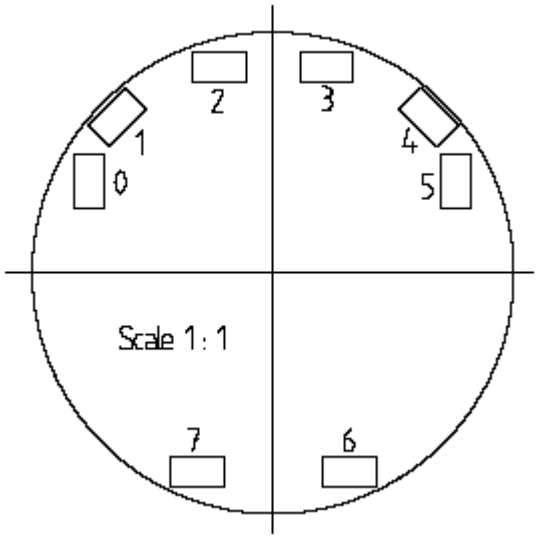


Figure 64: Position of the 8 IR sensors

To obtain a signal, first the sensors send out an infrared beam into the surroundings. If there is an obstacle (not dark color) present the beam is reflected off the object and received back to the sensor. This is the reflected value of the sensor. But the beam can also be sent into the surroundings to obtain a reading on the lighting in the area. When a beam is sent out without having a reflection it provides the ambient light reading.

Trial I

No obstacles present on the table

Sensor Number	Reflected Value	Ambient Light Value
0	36	496
1	60	496
2	36	500
3	56	500
4	88	500
5	28	500
6	64	496
7	104	496

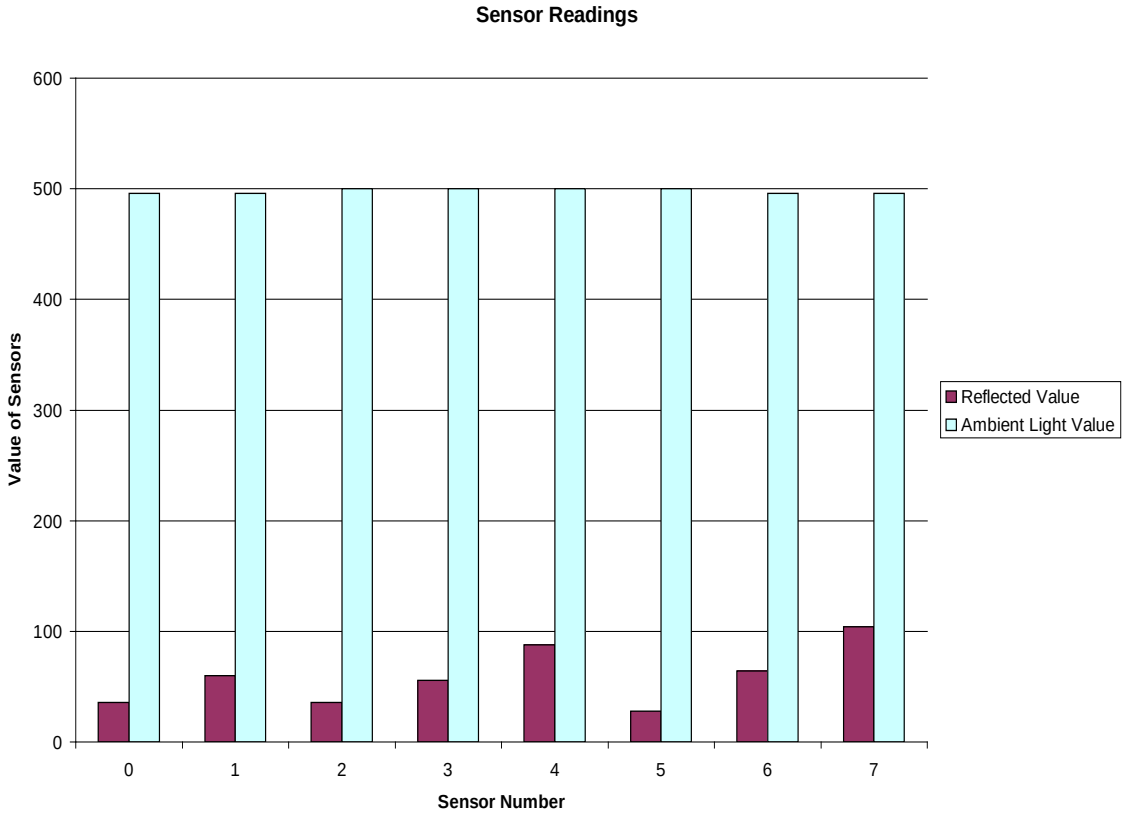


Figure 65: Trial I

Trial II
Object was placed very close to sensors 2 and 3. It was almost touching the robot

Sensor Number	Reflected Value	Ambient Light Value
0	36	496
1	72	496
2	1020	500
3	1020	496
4	100	496
5	28	496
6	68	496
7	112	496

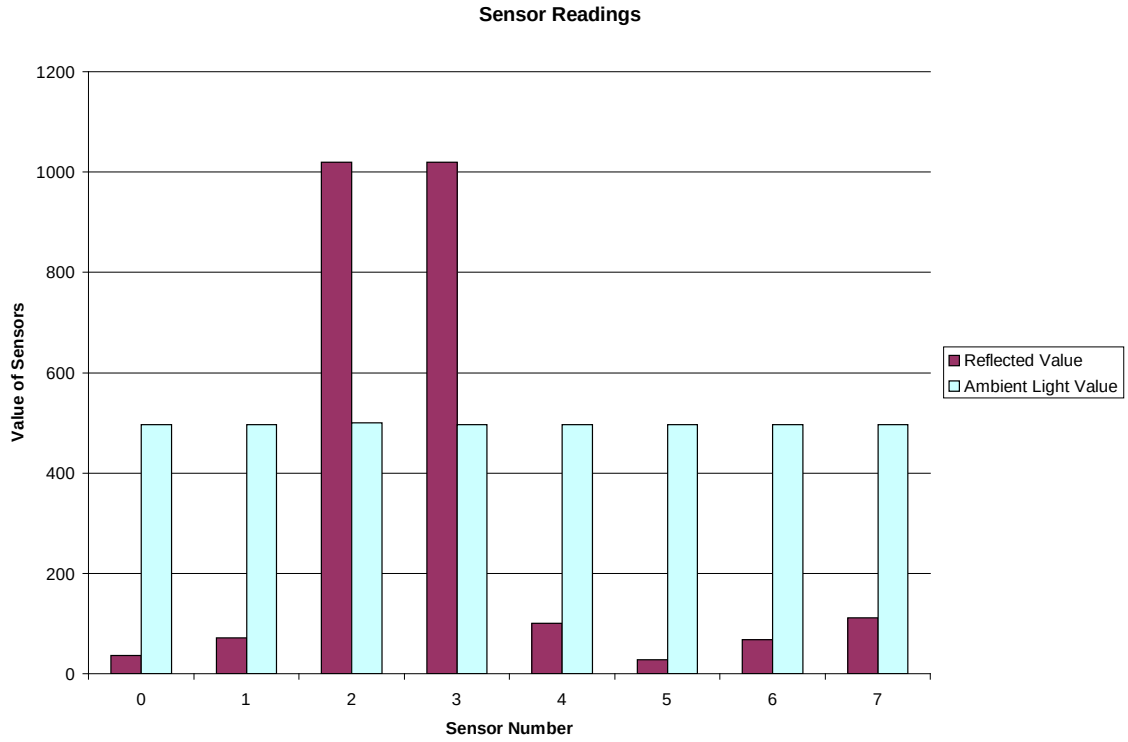


Figure 66: Trial II

Trial III
Object was placed 2-3 cm away from Sensors 2 and 3.

Sensor Number	Reflected Value	Ambient Light Value
0	40	496
1	72	496
2	116	500
3	148	500
4	88	496
5	28	500
6	68	500
7	112	496

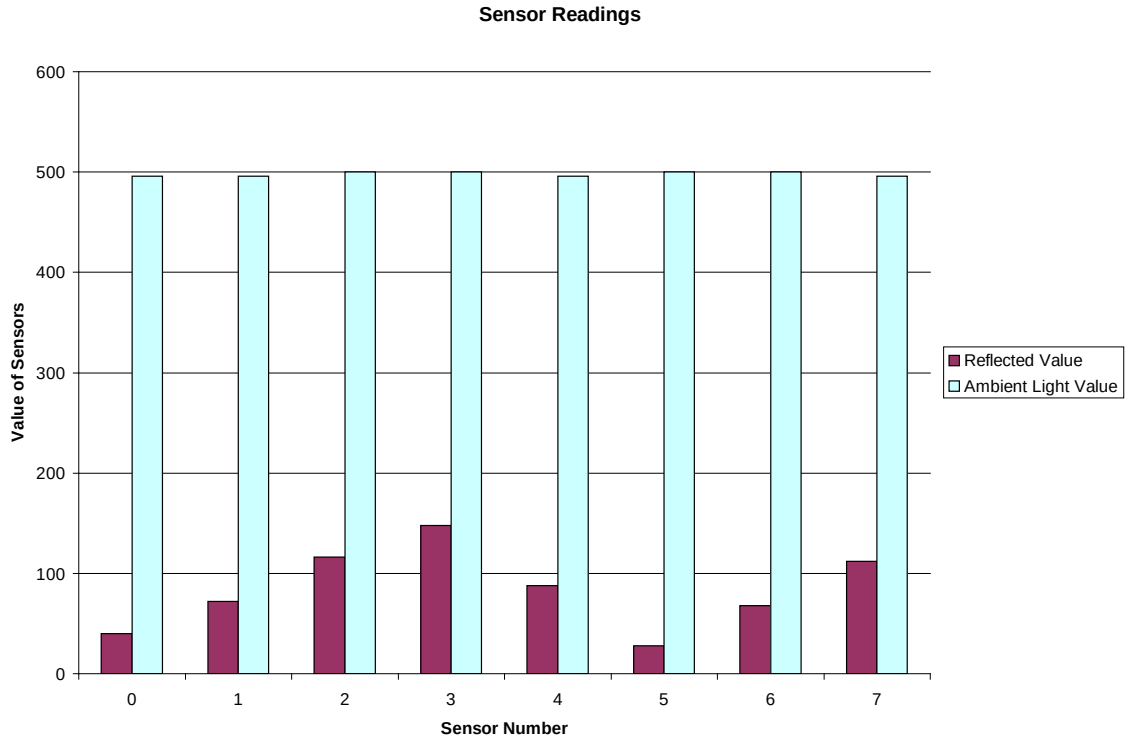


Figure 67: Trial III

Trial IV

Object is 5cm away

Sensor Number	Reflected Value	Ambient Light Value
0	36	496
1	68	496
2	60	500
3	80	500
4	92	496
5	32	500
6	68	496
7	116	496

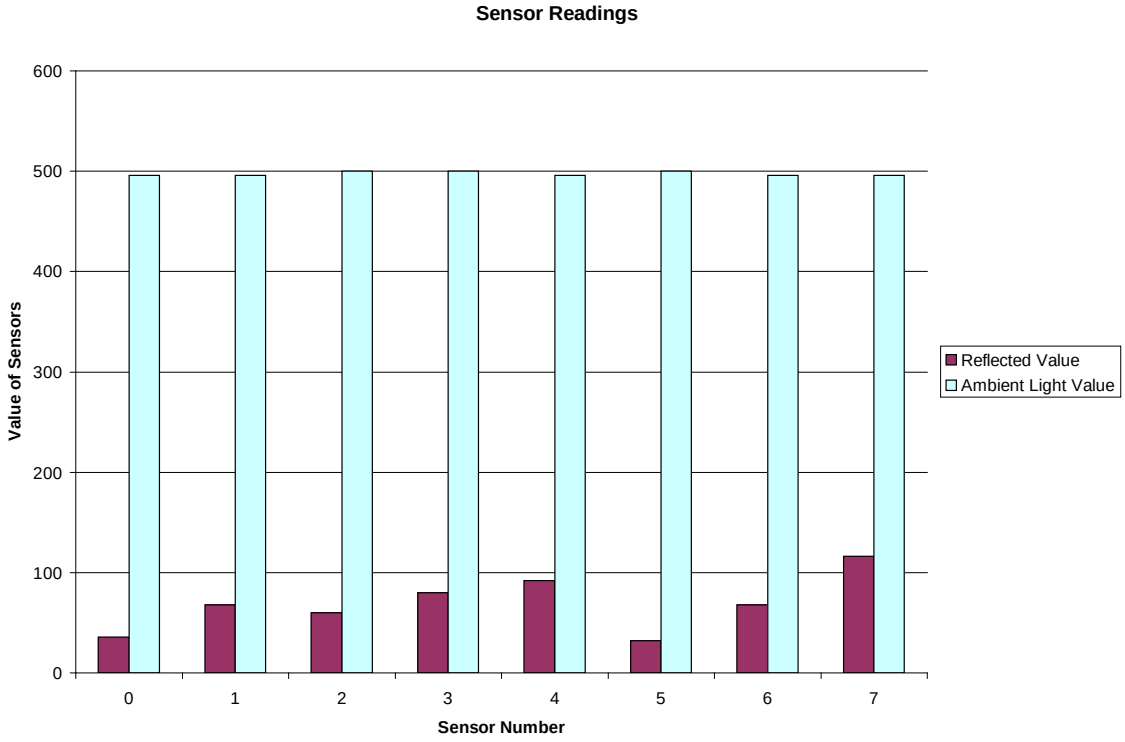


Figure 68: Trial IV

Trial V
Object is 8 cm away

Sensor Number	Reflected Value	Ambient Light Value
0	36	496
1	68	496
2	44	500
3	60	500
4	84	496
5	32	500
6	68	500
7	112	496

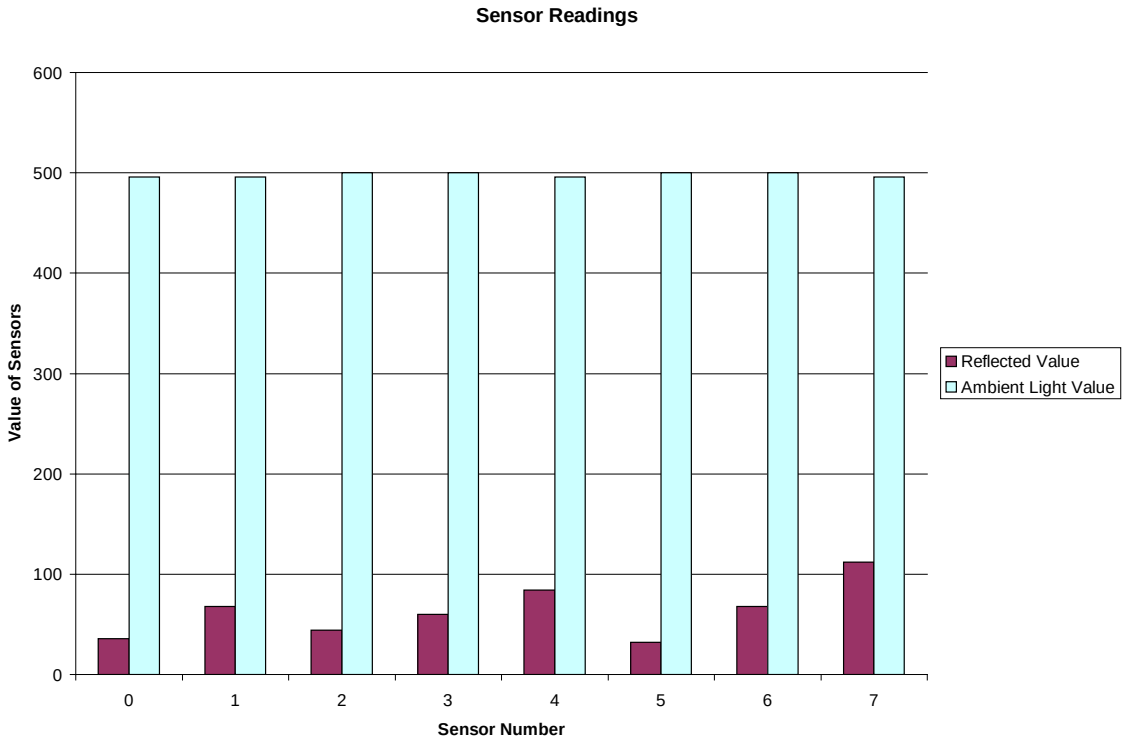


Figure 69: Trial V

Trial VI
Object is 15 cm away from the robot

Sensor Number	Reflected Value	Ambient Light Value
0	36	496
1	68	496
2	36	500
3	56	500
4	84	500
5	28	500
6	72	496
7	108	496

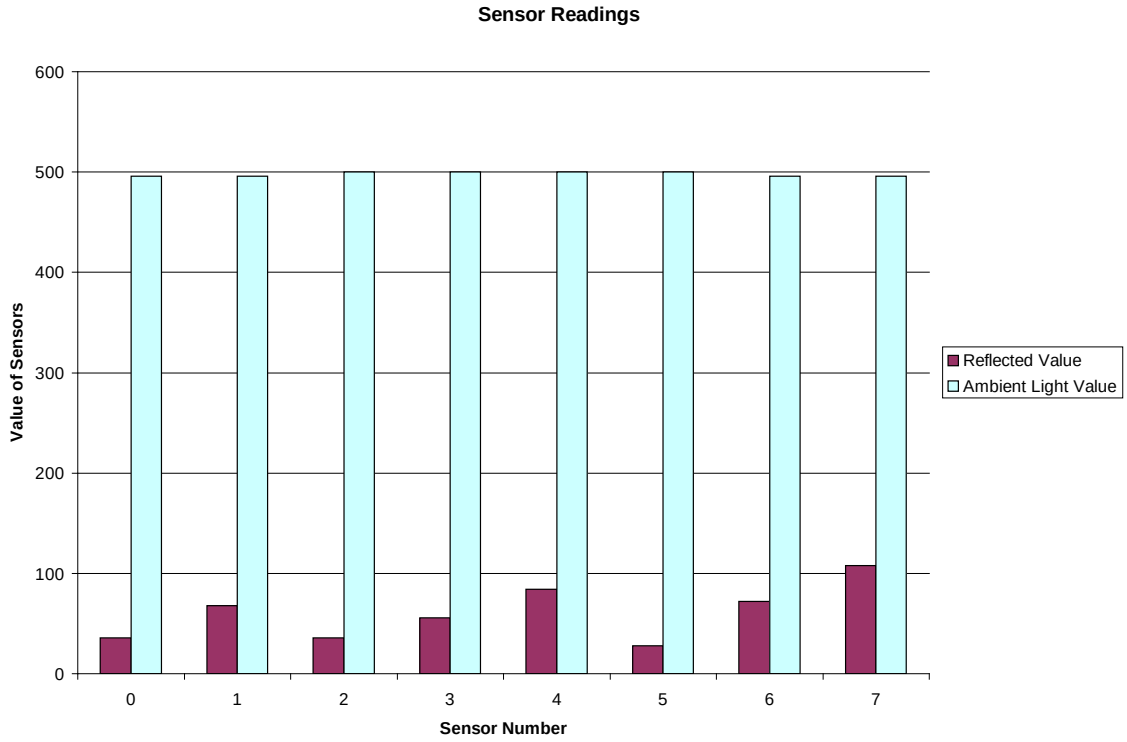


Figure 70: Trial VI

Analysis

The ambient light values are the general value of the IR sensor used to detect the light in the surrounding areas of the robot. The darker the room the higher the number will be, with a maximum value of 500. It can be noted from above that the ambient value is always around 500 although the tests were conducted in a room with average light where the source of light was florescent tube lights. One possible explanation is that the robot's shadow might be interfering with the sensors. Alternatively, since there is no variation in the light the values are always high.

Conclusion

Based on the values it was concluded that the ambient light readings cannot be used to detect obstacles or objects that might be in the vicinity of the robot. Instead the IR sensors reflected readings can be used. But to use the values they have to be calibrated first since the readings are not 0 when an object is not present (see Trial I). Thus, if any sensor has a reading of about 150 or greater then it indicates that an object is present. Also, the robot is not able to detect objects that are placed greater than 5cm away. Beyond this point the values can change arbitrarily without providing accurate results.

Scenario Testing and Evaluation

Section 3.4: Scenario Testing and Evaluation
Scenario I – Finding the baby

This is a simple case where the robot can easily find the baby. This means that the baby is initially in the line of sight of the robot so that the robot can easily find the child with out moving around. There are no obstacles placed around the room during this test. During this scenario the baby was present in the room 5 out of the 10 times this test was run. Each time the test was conducted the position and direction of the robot was changed. Given below in Fig. 71 are the images with the approximate positions of the robot and the baby during each of the tests. During the first 5 trials the baby was not present, and then the baby was placed in different locations around the room. Also, below in Fig. 72 is a photo taken from the far end of the table at the start of one of the tests.

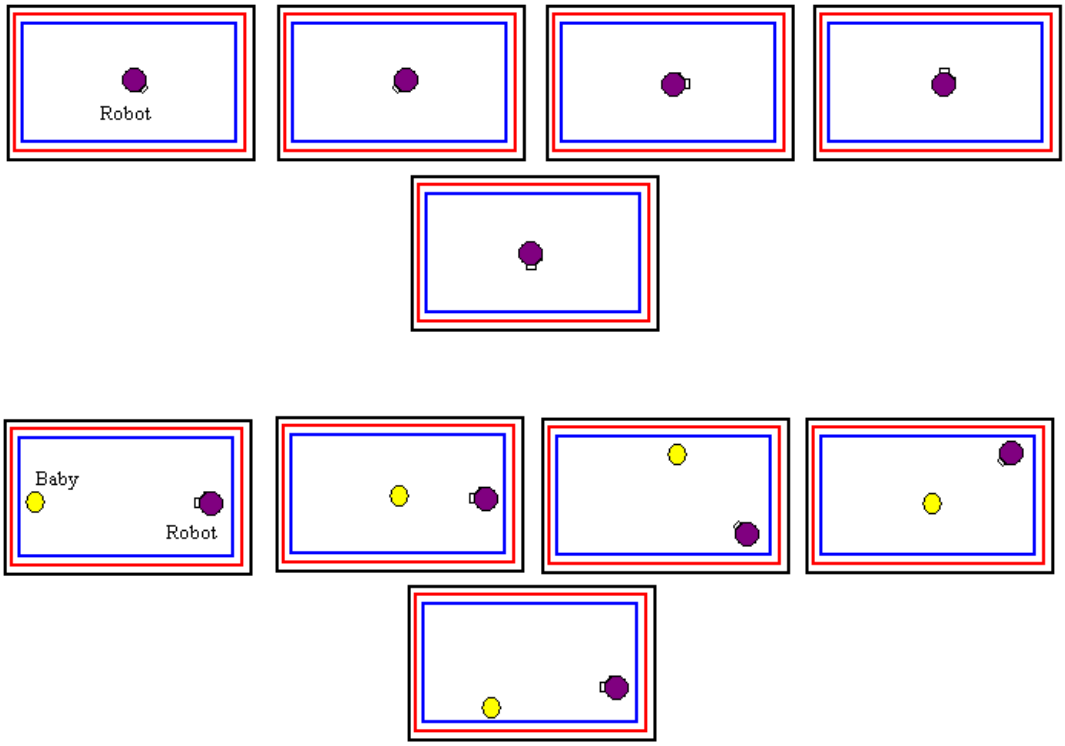


Fig: 71: Top view

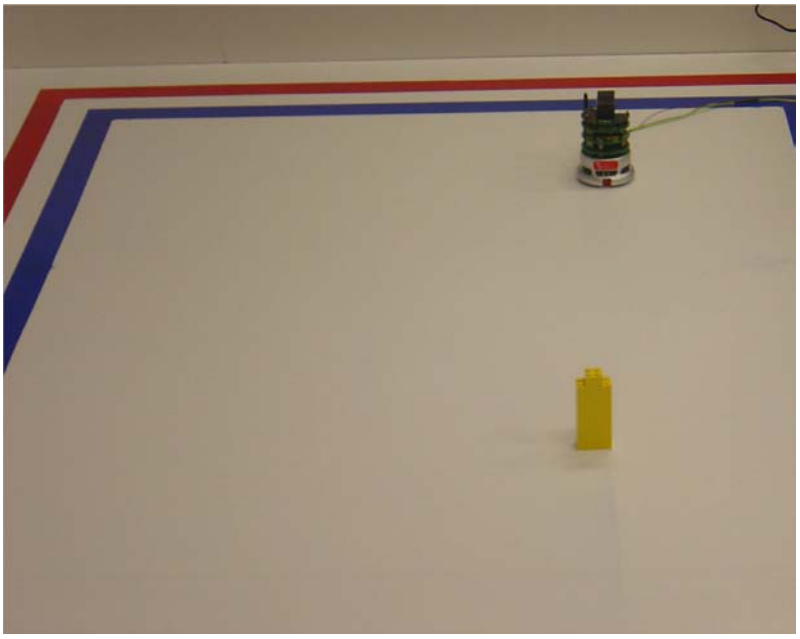


Figure 72: End view

Result

1. Scenario I

	Baby found	Baby not found
No baby present	0	5
Baby present	4	1

$$Success\_percentage = \frac{4+5}{10} \times 100 = 90\%$$

Scenario II – Simplified Obstacle Detection and Avoidance

This setup is of an intermediate difficulty level. Thus, even after the robot has located the child there is an obstacle that the robot has to avoid in order to get to the child. The rectangular box shaped obstacle with dimensions 10 cm long, 5 cm high and 0.9 cm wide was placed between the robot and the child. Also, the robot has to use the camera to detect whether the obstacle is present or not. The test was performed 5 times with the obstacle present and 5 times with the obstacle absent. The obstacle was removed to establish a ground truth to detect if the robot gives false readings. Each time the experiment was performed the locations stayed the same as shown in Fig. 73. During each of the tests it was important to determine whether the robot maintains it course or deviates from it. This determines whether the robot will reach the baby after it has passed around the obstacle. Thus, this situation was also tested in this scenario. Once the robot has successfully avoided the obstacle it still needs to determine the baby's location in the room and reach it successfully. Thus, sometimes the baby was placed slightly to the left or right of the robot instead of directly in front of it. Also, below in Fig. 74 is a photo taken from the side of the table at the start of one of the tests.

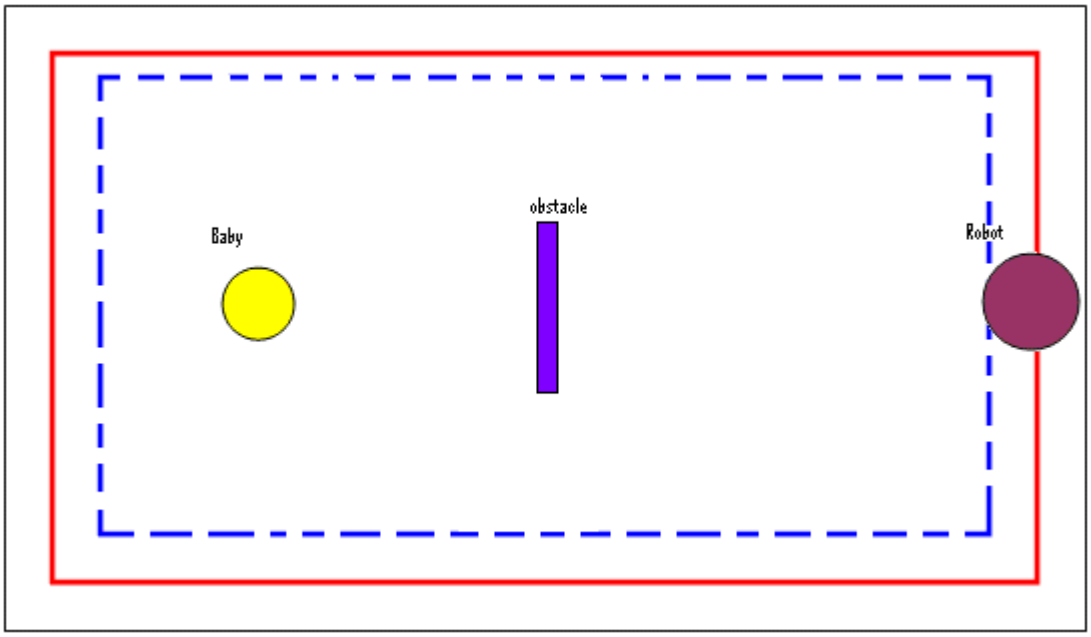


Fig 73: Obstacle present

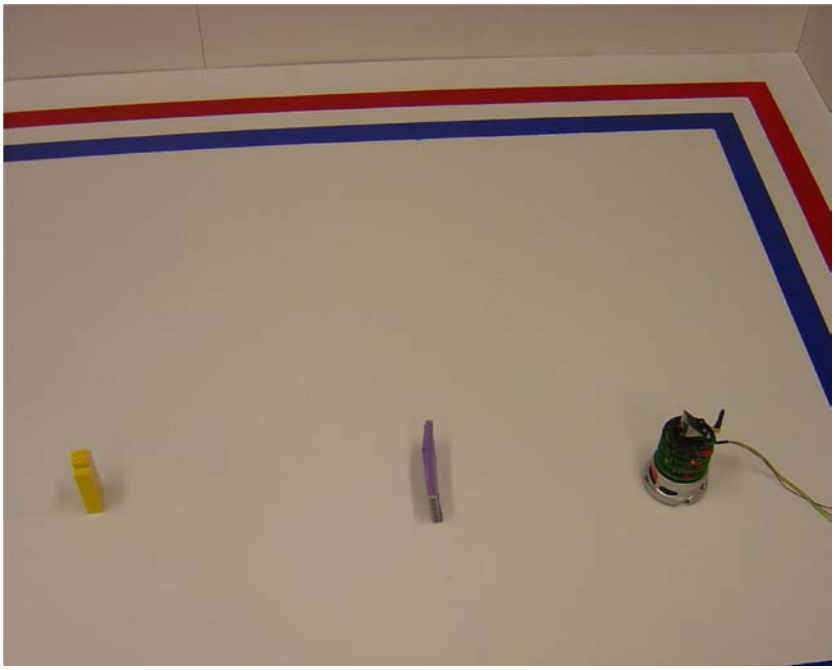


Figure 74: Side view

Result

A. Scenario II

	Obstacle found	Obstacle not found
Obstacle present	5	0
Obstacle absent	2	3

$$Success\_percentage = \frac{5+3}{10} \times 100 = 80\%$$

B. Scenario II

	Reached baby	Didn't reach the baby
Obstacle present	5	0
Obstacle absent	3	2

$$Success\_percentage = \frac{5+3}{10} \times 100 = 80\%$$

Scenario III- Complex Environment

This is the most difficult situation for the robot. The obstacles were placed in such a way that the robot had to scrutinize the pictures carefully to find the child. Then the obstacles were placed so that a complex solution is determined to reach the baby. Also, the structure of the obstacles is more composite as compared to the previous obstacle. Each time the test was run the location of the obstacles stayed the same. But sometimes the obstacle exactly in front of the child was switched with another of a different size and shape. Fig. 75 is a sample of one of the test variations. Two out of the 5 times, obstacle 3 was in front of the baby. Two times obstacle 2 was present and finally once obstacle 1 was present. The rest of the five times the test was run, there was no obstacle between the robot and the baby. Below in Fig. 76 is a photo taken from the side of the table at the start of one of the tests

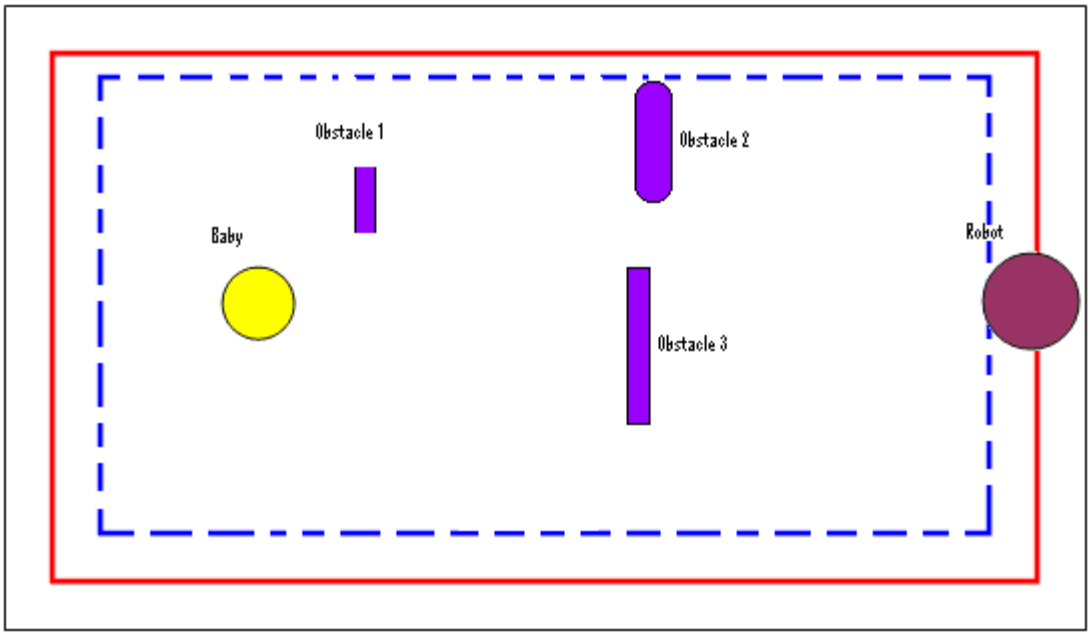


Figure 75: Complex environment

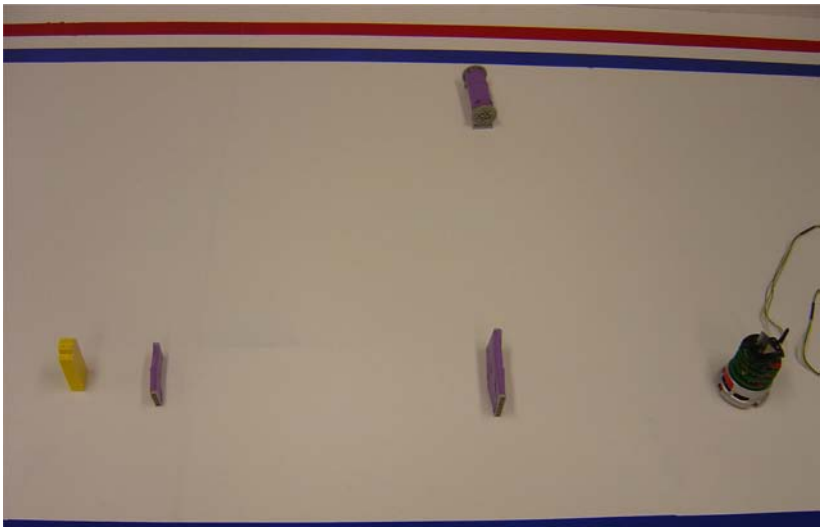


Figure 76: Side view

Result

A. Scenario III

	Found Obstacle	Obstacle not found
Obstacle present	4	0
Obstacle absent	1	5

$$Success\_percentage = \frac{4+5}{10} \times 100 = 90\%$$

B. Scenario III

	Reached Baby	Didn't reach the baby
Obstacle present	4	0
Obstacle absent	5	1

$$Success\_percentage = \frac{4+5}{10} \times 100 = 90\%$$

Analysis

During the first scenario the baby was found successfully when it was present. It was noted that if the baby is more than 60 cm away from the robot, the camera cannot detect any objects that are present since the objects in the image become out of focus.

For scenario II and III only a simplified obstacle avoidance pattern was used. So the robot simply moves around the obstacle by assuming that the largest obstacle has been found. Other avoidance techniques might allow the robot to turn around the obstacle faster but would require excessive computation that might be unnecessary. The two motors present on the robot do not always move at the same speed. The left motor seems to turn faster than the right motor all the time. This causes the robot to obtain inaccurate results regarding the distance traveled across the table. Also, the low quality camera creates difficulties for the robot to distinguish between the color of the obstacle and the blue or

red borders. This causes the robot to sometimes assume that the baby or the boundary lines is an obstacle causing the robot to go around the baby instead of following it around the room.

The robot can successfully detect more than one obstacle present in its path. But this depends on the number of pictures it takes while looking for the baby. BabyBot is also able to determine whether the obstacle is present in front of the baby or behind it. But since these results are not very consistent they were not used during the testing.

Section 3.5: Autonomous Behavior

For the robot to display artificial intelligence three other behavioral scenarios were tested.

Behavior I – Search Path with No Obstacles

Fig. 77 illustrates the motion of the robot over an obstacle free table. It has been divided up into various sectors for purpose of easy illustration. The black boundary indicated the area on the table that is out of bounds to the baby. However, the two imaginary lines that run along the length of the table are for position reference only. The circle marked BB is BabyBot. The solid black arrows indicated straight line motion; while the curved arrows indicate turns in the direction of the arrows. Also, the purple asterisks indicate at which spots the robot stops to take photographs. During this test the baby was placed around room 6 out of the 10 times. Twice the baby was placed in Row1, once it was placed in Row2 and once in Row3. This test was conducted to determine if the robot maintains the course while searching for the baby around the room.

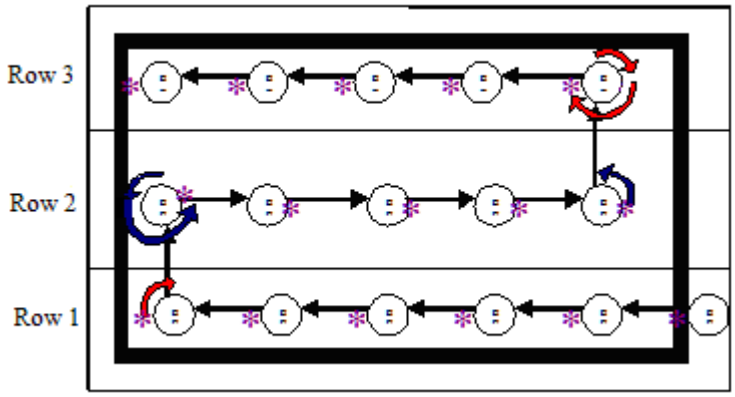


Figure 77: Search path of BabyBot

Result

Behavior I

	Baby found	Baby not found
No baby present	1	3
Baby present	5	1

$$Success\_percentage = \frac{5+3}{10} \times 100 = 80\%$$

Behavior II – Distraction and Alarm

Once the robot finds the baby it should follow it around the room. If the baby gets close to the blue line the robot should send a signal to distract the baby with the help of the flashing LEDs. Also, if the baby gets into the out of bounds area the robot should send another signal and activate the alarm to alert the parents. When determining whether the robot sends the signals in time, the robot was sometimes moved close to the horizontal blue line and the rest of the time the baby was moved closer to the vertical blue line at the end of the table. These lines are illustrated in Fig. 78 below.

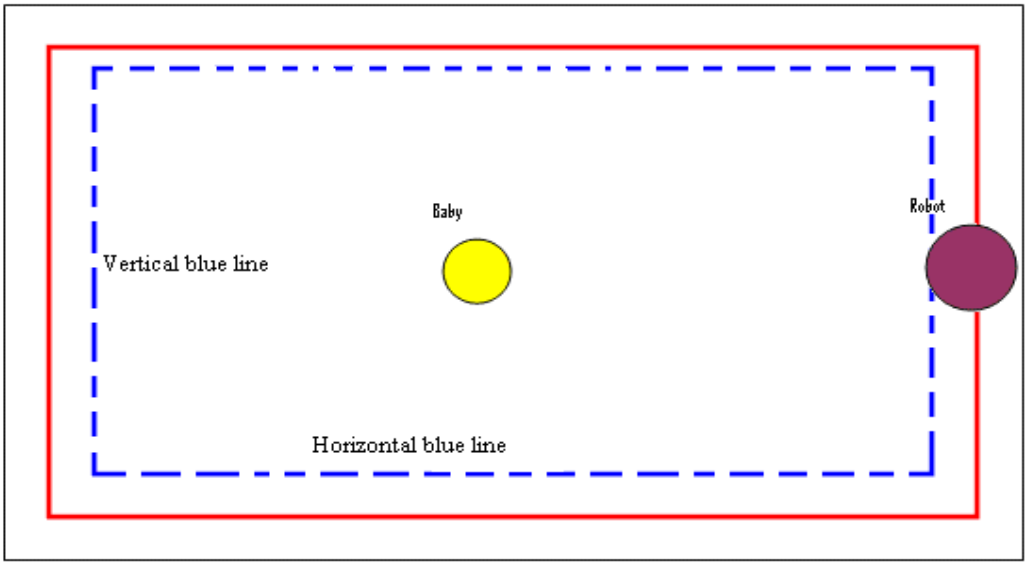


Figure 78: Line indications

Result

A. Lights

	Lights on	Lights off
Within range	3	0
Out of range	2	5

$$Success\_percentage = \frac{5+3}{10} \times 100 = 80\%$$

B. Alarm

	Alarm on	Alarm off
Within range	4	1
Out of range	1	4

$$Success\_percentage = \frac{4+4}{10} \times 100 = 80\%$$

C. Radio Signal (to flash lights)

	Sent signal	No signal sent
Within range	3	0
Out of range	2	5

$$Success\_percentage = \frac{5+3}{10} \times 100 = 80\%$$

Behavior III – Trapped

When the robot is following the baby around the room it needs to obtain signals from the IR sensors to determine the next course of action. If the baby is on the left side (close to sensors 0,1) of the robot, BabyBot should turn left to face the baby. Similarly it should turn right if the baby is on the right side (close to sensors 4,5). Also if the baby gets too close to the robot, then the robot should move back to get away from the baby. But if it gets trapped between the baby and a wall or an obstacle it will send a signal to activate the alarm that notifies the parents. Shown below in Fig. 79 is a possible situation when the robot can get trapped between the baby and the obstacle. If the baby moves closer to the robot, the robot will have to move backwards causing it to get close to the obstacle. When the back sensors (sensors 6,7) have a reading, indicating that an object is present behind the robot, the robot will sound the alarm. This occurs when the robot is about 2-3cm in front of the obstacle and less than 5cm away from the baby.

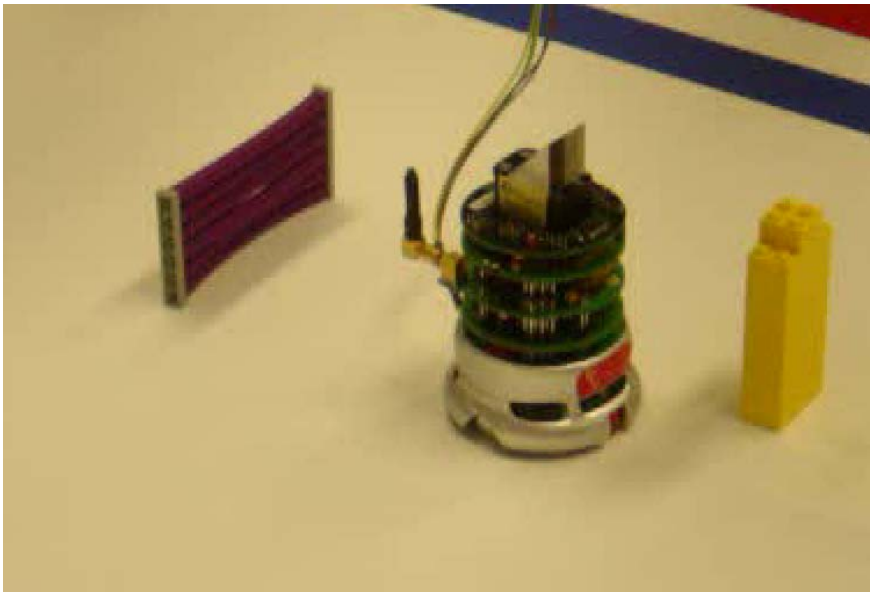


Figure 79: Setup for getting trapped

Result

A. Trapped

	Signal sent	No signal sent
Trapped	5	0
Not Trapped	0	5

$$Success\_percentage = \frac{5+5}{10} \times 100 = 100\%$$

B. Following baby around room

	Baby on right side	Baby not on right side
Turned right	4	1
Didn't turn right	0	5

$$Success\_percentage = \frac{4+5}{10} \times 100 = 90\%$$

C. Following baby around room

	Baby on left side	Baby not on left side
Turned left	4	1
Didn't turn left	0	5

$$Success\_percentage = \frac{4+5}{10} \times 100 = 90\%$$

D. Following baby around room

	Baby in front of robot	Baby not in front of robot
Stopped in front of baby	4	0
Hit the baby	1	5

$$Success\_percentage = \frac{4+5}{10} \times 100 = 90\%$$

Analysis

Behavior I- It was determined after the experiments that the robot will maintain the search path successfully if there are no obstacles present in the room. One important consideration during this type of test is that the serial cable should not be dragging on the table. If it does then the robot will veer towards the right since the added weight of the cord pulls on the robot turning it to the right. Thus, while the tests were being performed the cord was held up and away from the table. If there are obstacles present this search path will not work since the robot will not move the same distance each time. Hence there is a chance that the robot might run into the wall if it travels too long in one direction while avoiding obstacles.

Behavior II - This behavior of the robot was the most difficult to replicate successfully. The distance traveled by the robot depends on the angle on which it is moving along the table. This can dramatically change the results because if the angle is greater than 90 degrees sometimes negative values are obtained for the vector representation of the x and y coordinates. It was important to keep track of the angle each time the robot turns. The most successful tests were when the robot was moving along the bottom of the table or when it reached the end of the table. When the robot turned around 180 degrees, false results were obtained and the robot sent a signal to activate the alarm earlier than expected.

Behavior III - It must be noted that the IR sensors are very sensitive at close ranges. Sometimes while moving the baby around the robot, BabyBot detects fingers then assumes they are the baby. This causes the robot to sometimes get confused and turn in the wrong direction. Also, if the baby is not directly aligned with sensors 2, 3(front of robot), sensors on the either side of the robot (sensors 1 or 4) might get a reading. This causes BabyBot to turn back and forth since it is stuck in the loop and continuously turns left and right until the baby moves to another location.

When backing away from the baby the robot moves backwards slowly (reading sensor 6, 7 values). The reason for this is that it is more important to keep track of the baby than to get away from it. This was done to ensure that the robot does not lose track of the baby but also does not get picked up by the child.

Pseudo Codes

Section 3.6: Pseudo Codes
Finding the baby

Initialize the necessary variables
Send a message to the camera turret to take a picture
Receive the acknowledgment that the picture was taken successfully

While going down the rows
    Read two rows at a time and store into two matrices of size 160
    Check the values of the red, green and blue values of a pixel
        If a yellow pixel is found print information and jump out of the loop
            Else go to the next column in that row
If all 120 rows have been checked then print that the baby has not been found

Obstacle Avoidance

Initialize the necessary variables
Send a message to the camera turret to take a picture
Receive the acknowledgment that the picture was taken successfully

While going down the rows
    Read two rows at a time and store into two matrices of size 160
    Check the values of the red, green and blue values of a pixel
        If a purple pixel is found print information and jump out of the loop
            Else go to the next column in that row
If all 120 rows have been checked then print that there is no obstacle is present in the path

If an obstacle is present move forward until one of the IR sensors detect the obstacle (value>150)
Then turn right and move forward for 20cm(max length of the biggest obstacle)
Turn left and move forward 10cm(max width of the biggest obstacle)
Turn left and go forward 20cm again to reach the same distance as started before the obstacle was reached
Turn left to face in the same direction as was before the obstacle was reached

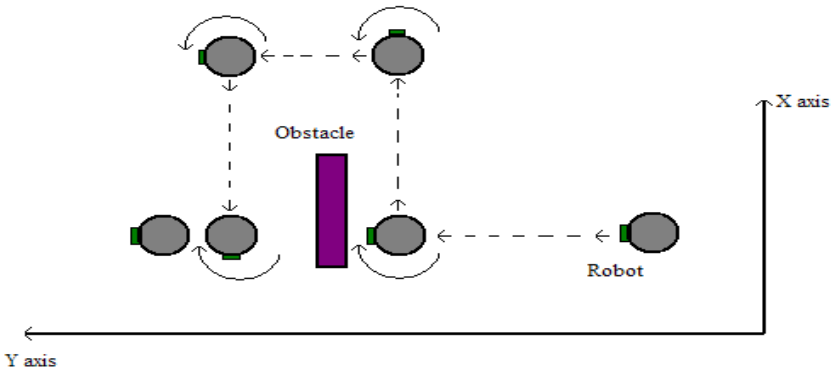


Figure 80: Movement of the robot around the obstacle

Search Path

Initialize the necessary variables
Set n=1
Depending on the n value (<27)
    Move forward for about 27cm
    Stop and take a picture and process it
    If no objects have been found then repeat
    If n =8,10,17 or 19
        Turn right(8 and 10) or left (17 and 19)
After each movement and picture increment n and repeat the tasks
If n>27 then sound the alarm that baby has not been found

Following the baby around the room

Once reached the baby keep checking the IR sensor values

If any sensor values >150 stop since robot is close enough to the baby
    If the left sensors have a higher reading(>500)
        Turn left to face the baby
    Else if the right sensors have a higher reading(>500)
        Turn right to face the baby
    If the front sensors have a higher reading(>500)
        Move backwards to move away from the baby a little
        If the back sensors have a reading(indicating a wall)
            Sound the alarm (alert parents) since trapped nowhere to move
            Exit from the program
Else move forward towards the baby

Distraction Techniques

Initialize the x and y position of the robot in the beginning of the program
Keep a running tally of movements along the table (increment x or y accordingly)
Once reached the baby and there are no obstacles surrounding it
    Determine when the robot moves forward, store value into a variable a
    If the robot turns right increment the angle by about 35 degrees
    Else if the robot turns left then decrement the angle by 35 degrees
    At the end determine the new x and y position of the robot using vectors
        Y = a*cos(angle)
        X=a*sin(angle)
    If the X or Y positions correspond to the area of the blue
        Send a signal to the radio base and turn on the lights
    Else if the baby is in the out of bounds area
        Send a signal to radio base and turn on the alarm and alert the parents
        Exit the program
    If the angle>=360 return value back to 0
If no change then keep following the baby around the room and keep checking the X and Y coordinates of the baby

SECTION IV

Evaluation and Recommendations

Evaluation

Section 4.1: Evaluation

Upon review of the entire project it can be determined that the Khepera II robot is an excellent tool for learning the different aspects that are needed for designing and creating an effective robotic system.

Image processing is an important field that has gained interest in the past few years. The filtration method used for this project can be modified so that the robot can detect different colors along with the two that it detects right now.

The robot has an average success rate of 83.33% proving that this prototype system has successfully accomplished all tasks as required. Certain tasks can still be modified to obtain a greater success rate but this would require additional computation and testing that require additional time and resources.

Recommendations

Section 4.2: Recommendations

The content and amount of research needed for this project was at times overwhelming. Understanding the terms and concepts that pertain specifically to the Khepera II robot are extremely important. This helps to understand the complex components of the robot and their limitations.

In the future it would be recommended to read the robot and turret manuals as well as learn how to setup the robot before beginning the senior design project. This would be a good method for time allocation that would allow students to focus on the design aspects of the project instead of the basis understanding of robotic parts.

Also, the compatibility of the robot with other sensors or devices is extremely important for making modifications to any design. Thus it would be a good idea for students to understand the basic layout of the robot and the turret connections. This might give them a better idea of the communication method used between the robot and the turrets. This would also allow them to find other components that might be more suited to the method.

Conclusion

At the on set of this project it was proposed that a robotic system would be developed to help parents monitor a child. We set out to accomplish the tasks of finding the baby, following the baby, distracting the baby and notifying the parents when necessary. All of these items have been developed and tested. Over the course of the building process there have been several changes in the methods used to accomplish each of the individual tasks. Ultimately with the evaluation and testing of various methods the most efficient methods were chosen.

The algorithms generated allow the robot to autonomously navigate around the testing environment while performing all the required tasks. All the components used helped to make the robot more intelligent and aware of the environment. Additional hardware was designed to display the results of this artificial intelligence design. Understanding the software was also an important aspect of robotic controls design. Due to the versatility of the Khepera II robot, it could have been programmed using different programming languages, such as C, LabVIEW or MatLAB. During this project C was used because of the ease with which the projects can be modified and debugged.

This robotic child monitoring prototype system has depicted how a Khepera II robot, along with certain components, can be used to monitor the movements of a child. Complex algorithms and components have been generated that take into consideration the different aspects of engineering, such as robotics, electronic design, artificial intelligence, and motor control applications.

References

1. The K-Team Web site www.k-team.com (last visited 04/17/2006)

2. Download a terminal emulator http://hp.vector.co.jp/authors/VA002416/teraterm.html (last visited 04/17/2006)

3. Electronic component data sheets. www.datasheetcatalog.com (last visited 04/19/2006)

Appendices

Appendix A: C program

Final code

This C program works for all the different scenarios that were tested for this project.

```
/* ECE 406
Senior Design II
Spring 2006

The BabyBot
By: Christole Griffith and Parul Reddy

This code allows for certain autonomous behavior of the robot.
It allows the robot to take pictures process then and determine
whether a baby or obstacle is present
Also, it moves around the obstacle, follows the baby around,
and sounds the alarm/lights when necessary*/

//Include necessary header files
#include <sys/kos.h>
#include        <stdlib.h>
#include        <stdio.h>
#include        <math.h>
#include        <stdbool.h>

static int32 vIDProcess[3];

/*
 * Process 0
 for all actions
 */
static void
process_0 ()
{
        //initialize the variables
        uint8  message[]={25,2,'I',0};  // this function is used to send the message to the camera turret to
read line 0 of an image.. an image must be acquired first
    uint8  message2[]={25,1,'Q'};  // this function is used to send a message to the camera turret to take a
picture
        uint8 message3[] = {4,19,'S',0,16,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}; // this function is used to send a
message from the radio turret to the radio base

        uint8 size3=22; // the proper size of message for the given radio communication
        uint8 size=4; // the proper size of message for the given camera 'I' command
    uint8 size2=3; // the proper size of message for the given camera 'Q' command

/*start of variables*/
        uint32 time=0;
        int32  status;
    uint8  line0[160],line1[160];
        uint32 sensor[8];
```

```
        uint32 newSensor[8];
        uint32 GreenPixel=0;

        bool find = FALSE;
        bool obstacle = FALSE;

        uint8 row, column;
        uint8 row1, column1;
        uint8 i,j,k,w;
        uint8 l=0;
        uint8 q=0;
        uint8 n=1;
        uint8 turn;

//      float angle=0.7853;
        float theta=0.0000;
        uint32 numAngle=0;
        uint32 x=5000;
        uint32 y=625;
        uint32 yInc, xInc,a,aNew,forward,backward;

        uint32 distance,ytotal,xtotal;
        uint32 distX, distY;
        uint32 leftSens[2], rightSens[2];
        uint8 leftturn1, rightturn1;
        uint8 leftturn=1;
        int8 rightturn=-1;
        uint32 position0, position1;
/*end of variables*/

  //continuously perform the process
   for (;;)
  {
        tim_suspend_task(1000);   //time set up function
        time++;  // variable
        mot_config_speed_1m (0, 3500,800, 100);    // bios command init motor 0
        mot_config_speed_1m (1, 3500, 800, 100);   //bios command init motor 1

        position0=mot_get_position(0); //set variables to the bios command reading
        position1=mot_get_position(1);// of the position counters on the motors
        k=0;

        mot_new_speed_1m (0, 6);//left motor, bios commands to set the
        mot_new_speed_1m (1, 6);// new speed of the robot

        forward=mot_get_position(1); // set variable to the bios command reading of the position counters
of the robot

        //if 27cm has been reached
        if(position0>=3400 && position1>=3400)
        {

                mot_stop();
                status = mot_put_sensors_1m(0,0);
                status = mot_put_sensors_1m(1,0);
                tim_suspend_task(1000);
```

```
        //Take a picture
        while(msg_reserve_channel(0))
           tim_switch_fast();

        status=msg_send_message(message2,size2);
        printf("msg Q sent (%d)\r\n",status);

status=msg_receive_message(line0,1);
        if(!status)
        printf("Recv: %c\n\r",line0[0]);

        msg_release_channel(0);
        tim_suspend_task (1000);
        i=0;
        j=0;

        //Analyze the picture 2 rows at a time
        while(j<100)
        {
                message[3]=j;
                status=msg_send_message(message,size);//send I
        //      printf("msg I sent (%i)\r\n",status);
                if(!status)
                        status=msg_receive_message(line0,1);
                //read line0
                if(!status)
                        status=msg_receive_message(line0,160);

                tim_suspend_task (1000);

                //then read line1
                j++;
                message[3]=j;
                status=msg_send_message(message,size);
        //      printf("msg I sent (%i)\r\n",status);
                if(!status)
                        status=msg_receive_message(line1,1);
                if(!status)
                        status=msg_receive_message(line1,160);

                i=0;
                //read all the columns of the 2 rows
                while(i<160)
                {
                        if(!obstacle)
                        {
                                //find obstacle
                                if(line1[i]>=47 && line1[i]<=65)//green
                                {
                                        if(line1[i+1]>=44&& line1[i+1]<=50)//blue
                                        {
                                                if(line0[i]>=42 && line0[i]<=46)//red
                                                {
                                                //this means there is an obstacle
```

```
                                                row1=j;
                                                column1=i;
                                                printf("obstacle at row = %i,column= %i
\r\n",row1, column1);
                                                printf("R=%i, G=%i,
B=%i\r\n",line0[i],line0[i+1],line1[i+1]);
                                        //      j=120;
                                        //      i=160;
                                                obstacle=TRUE;
                                                }
                                        }
                                }
                                else{
                                row1=120;
                                column1=160;
                                }
                        }
                        //find baby
                        if(!find)
                        {
                                GreenPixel=line1[i];
                                if(GreenPixel>=110 && GreenPixel<=150)//GREEN
                                {
                                        if(line1[i+1]>=50 && line1[i+1]<=60)//BLUE
                                        {
                                                if(line0[i]>=95 && line0[i]<=100)//RED
                                                {
                                                        printf("Baby Found\r\n");

                                                        row=j;
                                                        column=i;

                                                        printf("row=%d,
column=%d\r\n",row,column);
                                                        printf("R=%i, G=%i,
B=%i\r\n",line0[i],line0[i+1],line1[i+1]);
                                                        var_change_led(0);
                                                        find=TRUE;
                                                //      j=120;
                                                }
                                        }
                                }
                                else
                                {
                                        row=120;
                                        column=160;
                                }
                        }
                        i=i+2;//to keep going along the row
                }
                if(obstacle && find)
                        j=120;//jump out of the loop

                j++;
        //      printf("In Row = %i\r\n",j);
        }
```

```
//if(find)
//          printf("baby\r\n");
msg_release_channel(0);
tim_suspend_task(500);
sens_reset();
y=y+forward;

printf("Y=%i\r\n", y);
//obstacle before the baby
while(obstacle)
{
        if(n<=17)
                turn=4;

        else
                turn =-4;
        //move forward
        mot_new_speed_2m(6,6);
        k=0;
        yInc=mot_get_position(1);
        //get the sensor reading
        while(k<8)
        {
                sensor[k]=sens_get_reflected_value(k);
                k++;
        }

        //obstacle present
        if(sensor[0]>=100 || sensor[1]>=150 || sensor[2]>=150 || sensor[3]>=150 ||
sensor[4]>=150)
        {
                printf("Going around obstacle\r\n");
                mot_stop();
                k=0;
                tim_suspend_task(500);
                //turn to the right
        //      tim_suspend_task(500);
                mot_new_speed_2m(-turn,turn);
                tim_suspend_task(1300);
                mot_stop();
                while(k<8)
                {
                        sensor[k]=sens_get_reflected_value(k);
                        k++;
                }
                //move forward along the length of obstacle
                while(distX<2500)
                {
                        mot_new_speed_2m(4,4);
                        tim_suspend_task(500);
                        distX=mot_get_position(1);
                }
                //stop and turn left
                tim_suspend_task(500);
```

```
                    mot_new_speed_2m(4,-4);
                    tim_suspend_task(1300);
                    mot_stop();
                    //move forward along the width of obstacle
                    while(distY<1250)
                    {
                            mot_new_speed_2m(4,4);
                            tim_suspend_task(500);
                            distY=mot_get_position(1);
                    }

                    //turn left
                    tim_suspend_task(500);
                    mot_new_speed_2m(turn,-turn);
                    tim_suspend_task(1300);
                    mot_stop();

                    distX=mot_get_position(1);
                    //move forward along the length of obstacle
                    while(distX<2500)
                    {
                            mot_new_speed_2m(4,4);
                            tim_suspend_task(500);
                            distX=mot_get_position(1);
                            //printf("Distance around the obstacle = %i",distX);
                    }


                    tim_suspend_task(500);
                    mot_new_speed_2m(-4,4);
                    tim_suspend_task(1300);
                    mot_stop();

                    row1=200;
                    obstacle=FALSE;

            }

}
//increment the total distance travelled in the y direction
msg_release_channel(0);
printf("Yinc=%i\r\n", yInc);
            y=y+yInc+distY;
printf("Ynew=%i\r\n", y);
yInc=0;
distY=0;
            //go to baby and follow it around the room
while(find && !obstacle)
{
            l=0;
            w=0;
            mot_new_speed_2m (3, 3);
            if(l==0)//get distance only once in the loop
                    aNew=abs(mot_get_position(1));
            else
                    aNew=0;
```

```
l=1;
k=0;
while(k<8)
{
	sensor[k]=sens_get_reflected_value(k);
	k++;
}

//baby detected
while(sensor[0]>=100 || sensor[1]>=150 || sensor[2]>=150 || sensor[3]>=150 ||
sensor[4]>=150
|| sensor[5]>=80 || sensor[6]>=150 || sensor[7]>=190)
{
	mot_stop();
	if(w==0)
	{
		//increment the x,y coordinates of the robot
		y=y+aNew*cos(theta);
		x=x+aNew*sin(theta);
		printf("aNew =%i\r\n",aNew);
		printf("y= %i\r\n",y);
		printf("x= %i\r\n",x);
		w=1;
	}
	l=1;
	aNew=0;
	w=1;
	k=0;

	//get updated sensor values
	while(k<8)
	{
		sensor[k]=sens_get_reflected_value(k);
		k++;
	}


	tim_suspend_task(1000);

	//turn left to face the baby
	if((sensor[0]>=100 || sensor[1]>=100) && (sensor[0]<=500 ||
sensor[1]<=500))
	{

		tim_suspend_task(500);
		mot_new_speed_2m(4,-4);
		tim_suspend_task(500);
		mot_stop();
		theta=theta-0.6108;
		if(theta>=6.283)
			theta=0;
		if(theta>=1.5707)
			theta = 3.14153-theta;
		aNew=0;
	}
```

```
//turn right
else if((sensor[4]>=100 || sensor[5]>=100) && (sensor[4]<=500 ||
sensor[5]<=500))
{
	tim_suspend_task(500);
	mot_new_speed_2m(-4,4);
	theta=theta+0.6108;
	if(theta>=6.283)
		theta=0;
	if(theta>=1.5707)
		theta = 3.14153-theta;
	tim_suspend_task(500);

	mot_stop();
}

//baby is too close
if((sensor[2]>=550 || sensor[3]>=550))
{
	//move backwards
	tim_suspend_task(500);
	mot_new_speed_2m(-2,-2);
	backward=mot_get_position(1);
	tim_suspend_task(500);

//	printf("Close to sensor 4,5\r\n");
	mot_stop();
	//baby too close and trapped
	if(sensor[6]>=200 || sensor[7]>=200)
	{
		while(msg_reserve_channel(0))
			tim_switch_fast();
		tim_suspend_task(100);
		status=msg_send_message(message3,size3);
		printf("Distract (%d)\r\n",status);

		//sound the alarm
		tim_suspend_task(5000);
		status=msg_send_message(message3,size3);
		printf("Distract1 (%d)\r\n",status);

		exit(0);//exit from program
}

}


//move back if baby is close on either side
if((sensor[0]>=550 || sensor[1]>=550) || (sensor[4]>=550 ||
sensor[5]>=550))
{
	tim_suspend_task(500);
	mot_new_speed_2m(-2,-2);
	backward=mot_get_position(1);
	tim_suspend_task(500);
```

```
                                        mot_stop();
                                }


                                        msg_release_channel(0);
                                //do the lights
                                if((y>=1625  && y<2750)|| (y>25500 && y<=26450) || (x>=1625  &&
x<2550)|| (x>=10056 && x<=10793))
                                {
                                        if(q==0)
                                        {
                                                while(msg_reserve_channel(0))
                                                        tim_switch_fast();
                                                tim_suspend_task(500);
                                                printf("Sending Lights\r\n");
                                                status=msg_send_message(message3,size3);
                                                printf("Lights (%d)\r\n",status);
                                                tim_suspend_task(500);
                                                msg_release_channel(0);

                                        }
                                        q=1;

                                //      exit(0);

                                }
                                //do alarm
                                else if(y>26450 || (y<1650) || (x>10793)|| (x<1625))
                                {
                                        while(msg_reserve_channel(0))
                                                tim_switch_fast();
                                        tim_suspend_task(500);
                                        printf("Sending Alarm\r\n");
                                        status=msg_send_message(message3,size3);
                                        printf("Alarm (%d)\r\n",status);
                                        tim_suspend_task(500);
                                        msg_release_channel(0);
                                        tim_suspend_task(2000);
                                        exit(0);
                                }

                        }

                }



                //If the baby has not been found
                n++;
                printf("In loop n = %i\r\n",n);

                //completed movements, turn in correct direction
                //to maintain the search path
                if(n==8 || n==10 || n==17 || n==19 )
                {
                        if(n==8)
```

```
                {
                        leftturn=7;
                        rightturn=0;
                }
                else if(n==10)
                {
                        leftturn=0;
                        rightturn=-7;
                }

                else if(n==17|| n==19)
                {
                        leftturn=0;
                        rightturn=7;
                }

                tim_suspend_task(800);
                printf("%d\r\n",n);
                mot_stop();
                tim_suspend_task(500);

                mot_new_speed_1m (0, leftturn);//left motor
                mot_new_speed_1m (1, rightturn);
                position0 =mot_get_position(0);
                position1=mot_get_position(1);

                tim_suspend_task(500);
                printf("position0=%i\r\n",position0);
                printf("position1=%i\r\n",position1);

                //after turned 90 deg stop
                if(position0>=300)
                {
                        mot_stop();
                        printf("Turning\r\n");
                        status = mot_put_sensors_1m(0,0);
                        status = mot_put_sensors_1m(1,0);
                        n++;
                }
                n++;
                printf("%d\r\n",n);

        }
    }

//if finished search path then stop
    if(n>=26)
    {
        mot_reset();
        mot_stop();
        while(msg_reserve_channel(0))
                tim_switch_fast();
        tim_suspend_task(100);
        status=msg_send_message(message3,size3);
        printf("exiting\r\n");
        tim_suspend_task(1000);
```

```
                    msg_release_channel(0);
                    exit(0);
            }
  }
  return 0;
}

int
main (void)
{
int32 status;

//initialze the process
       static char prName_0[] =
    "User 0 process, EF-98                    Rev. 1.00\r\n";

//reset robot controls
  com_reset ();
  var_reset ();
  sens_reset ();
  msg_reset();
  mot_reset ();                        /* initialize motors */
  tim_reset();

 status = install_task (prName_0, 800, process_0);
  if (status == -1)
    exit (0);

  vIDProcess[0] = (uint32) status;
  return 0;
}
```

Appendix B: Electronic Component Data Sheets

This appendix includes a datasheet for each of the following components

- **AND Gate**
- **Diodes**
- **Flip Flop**
- **LED**
- **Resistor and Capacitor**
- **Transformer**
- **Transistor**
- **5 pin Relay**
- **555 Timer**

Appendix C: Manuals

This appendix includes the manuals for each of the following robotic components

- **BIOS**
- **Camera Turret**
  - **VV6300 Sensor**
- **Khepera II**
- **Radio Base**
- **Radio Turret**